ACTS CKF Implementation

Rodrigo Estevam de Paula

April 1, 2024

Quick updates

Study and research

- Read chapters 1-5 of the Particle Detectors book
  - will proceed to chp 8-9
- Made a systematic bibliographic review (as part of a lecture course)
  - Returned 107 papers!
  - Next: update the research plan accounting for the systematic review
    - Due tomorrow (02.04.2024)

HGTD collaboration

- Will have a first meeting with them Thursday

Miscellaneous

- Conference to look up to: https://www.apsac.co/

Outline

- **How to add new algorithms into the ACTS framework**
  - Standard structure
  - How to integrate it into the python bindings

- **Implementation of Combinatorial Kalman Filter (CKF)**
  - Behaviour diagram
  - Code exploration
  - Verbose run

- **Extra: What to present to the HGTD simulation and performance team?**

Algorithms

ACTS algorithms

- Algorithms allow the examples framework to use the Core packages
  - The python bindings are included in the examples framework

```python
fitAlg = acts.examples.TrackFittingAlgorithm(
    level=customLogLevel(),
    ...,
    fit=acts.examples.makeKalmanFitterFunction(
        trackingGeometry, field, **kalmanOptions
    ),
    calibrator=calibrator,
)
s.addAlgorithm(fitAlg)
```

Example of algorithm usage in the python bindings

- Algorithms can be any part of the tracking processing chain, from simulation or propagation to seeding and reconstruction
- New ideas and methods are first implemented as algorithms and essential parts are later incorporated as a Core method
- Located at *<acts>/Examples/Algorithms*

[Add a new algorithm ACTS documentation page]

Algorithms structure

Has to inherit from IAlgorithm class

Config struct

execution function, inputs are read from ctx (context)

optional overheads to the execution

Input and output

```cpp
#pragma once

#include "ActsExamples/Framework/IAlgorithm.hpp"

#include <string>
#include <vector>

namespace ActsExamples {

/// Construct a user algorithm for demonstrator purposes
class UserAlgorithm final : public IAlgorithm {
 public:
  struct Config {
    /// Simple message
    std::string message = "Hello world";
  };

  /// Construct the user algorithm.
  ///
  /// @param cfg is the algorithm configuration
  /// @param lvl is the logging level
  UserAlgorithm(Config cfg, Acts::Logging::Level lvl);

  /// Run the algorithm.
  ///
  /// @param ctx is the algorithm context with event information
  /// @return a process code indication success or failure
  ProcessCode execute(const AlgorithmContext& ctx) const final;

  ProcessCode initialize() final;
  ProcessCode finalize() final;

  /// Const access to the config
  const Config& config() const { return m_cfg; }

 private:

  //Data input and output
  ReadDataHandle<SimSpacePointContainer> m_inputSpacePoints{this,
                                                            "InputSpacePoints"};
  WriteDataHandle<SimSeedContainer> m_outputSeeds{this,
                                                  "OutputSeeds"};

  Config m_cfg;
};

}  // namespace ActsExamples
```

Adding as Python binding

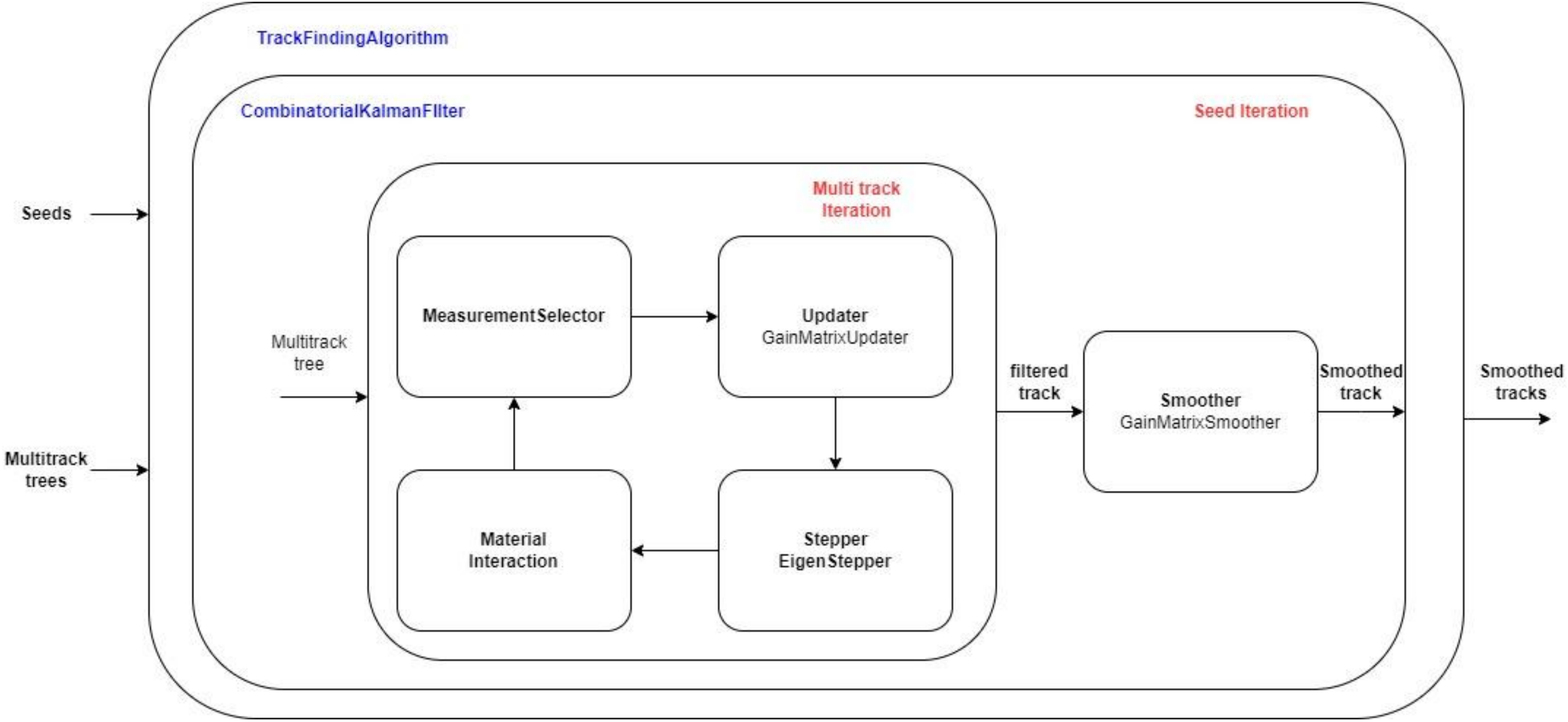
```
void addTruthTracking(Context& ctx) {
  auto mex = ctx.get("examples");

  ACTS_PYTHON_DECLARE_ALGORITHM(
      ActsExamples::TruthTrackFinder, mex, "TruthTrackFinder", inputParticles,
      inputMeasurementParticlesMap, outputProtoTracks);

  {
    using Alg = ActsExamples::TruthSeedSelector;
    using Config = Alg::Config;

    auto alg = py::class_<Alg, IAlgorithm, std::shared_ptr<Alg>>(
                   mex, "TruthSeedSelector")
                   .def(py::init<const Alg::Config&, Acts::Logging::Level>(),
                        py::arg("config"), py::arg("level"))
                   .def_property_readonly("config", &Alg::config);

    auto c = py::class_<Config>(alg, "Config").def(py::init<>());

    ACTS_PYTHON_STRUCT_BEGIN(c, Config);
    ACTS_PYTHON_MEMBER(inputParticles);
    ACTS_PYTHON_MEMBER(inputMeasurementParticlesMap);
    ACTS_PYTHON_MEMBER(outputParticles);
    ACTS_PYTHON_MEMBER(rhoMin);
    ACTS_PYTHON_MEMBER(rhoMax);
    ...
    ACTS_PYTHON_MEMBER(nHitsMin);
    ACTS_PYTHON_MEMBER(nHitsMax);
    ACTS_PYTHON_STRUCT_END();

    pythonRangeProperty(c, "rho", &Config::rhoMin, &Config::rhoMax);
    pythonRangeProperty(c, "z", &Config::zMin, &Config::zMax);
    pythonRangeProperty(c, "phi", &Config::phiMin, &Config::phiMax);
    pythonRangeProperty(c, "eta", &Config::etaMin, &Config::etaMax);
    pythonRangeProperty(c, "absEta", &Config::absEtaMin, &Config::absEtaMax);
    pythonRangeProperty(c, "pt", &Config::ptMin, &Config::ptMax);
    pythonRangeProperty(c, "nHits", &Config::nHitsMin, &Config::nHitsMax);
  }n go(f, seed, [])
}
```

- Interface has to be described at *<acts>/Examples/Python/src/*
- This example is present at *<acts>/Examples/Python/src/TrackFinding.cpp*
- Later the algorithm can be used with:

```
from acts .examples import *
help(addTruthTracking)
help(addTruthTracking)
```

Combinatorial Kalman Filter implementation

CKF behaviour diagram



EigenStepper - Numeric integration

```
Acts::Result<double> Acts::EigenStepper<E, A>::step(
    propagator_state_t& state, const navigator_t& navigator) const {

  // Chooses best step size (h) and calculate k1,k2,k3,k4 according to
  // the magnetic field
  ...

  // Update the track parameters according to the equations of motion
  state.stepping.pars.template segment<3>(eFreePos0) +=
      h * dir + h2 / 6. * (sd.k1 + sd.k2 + sd.k3);
  state.stepping.pars.template segment<3>(eFreeDir0) +=
      h / 6. * (sd.k1 + 2. * (sd.k2 + sd.k3) + sd.k4);
  (state.stepping.pars.template segment<3>(eFreeDir0)).normalize();

  if (state.stepping.covTransport) {
    state.stepping.derivative.template head<3>() =
        state.stepping.pars.template segment<3>(eFreeDir0);
    state.stepping.derivative.template segment<3>(4) = sd.k4;
  }
  ...
}
```

$$\vec{r}_{n+1} = \vec{r}_n + h\vec{T}_n + \frac{h^2}{6}(k_1 + k_2 + k_3).$$

$$\vec{T}_{n+1} = \vec{T}_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

Gain Matrix Updater - Filter

```
std::tuple<double, std::error_code> GainMatrixUpdater::visitMeasurement(
     InternalTrackState trackState, Direction direction,
     const Logger& logger) const {

  ...

  const auto H = trackState.projector
                     .template topLeftCorner<kMeasurementSize, eBoundSize>()
                     .eval();

    ACTS_VERBOSE("Measurement projector H:\n" << H);

    const auto K = (trackState.predictedCovariance * H.transpose() *
                    (H * trackState.predictedCovariance * H.transpose() +
                     calibratedCovariance)
                        .inverse())
                       .eval();

    ACTS_VERBOSE("Gain Matrix K:\n" << K);

    if (K.hasNaN()) {
      error = (direction == Direction::Forward)
                  ? KalmanFitterError::ForwardUpdateFailed
                  : KalmanFitterError::BackwardUpdateFailed;  // set to error
      return false;                                           // abort execution
    }

    trackState.filtered =
        trackState.predicted + K * (calibrated - H * trackState.predicted);
    trackState.filteredCovariance = (BoundSquareMatrix::Identity() - K * H) *
                                    trackState.predictedCovariance;
    ACTS_VERBOSE("Filtered parameters: " << trackState.filtered.transpose());
    ACTS_VERBOSE("Filtered covariance:\n" << trackState.filteredCovariance);
...
}
```

C(n) (covariance matrix) used instead of P(n) (error covariance matrix) !

Projection matrix **H**

$$\boldsymbol{K}^o(n) = \boldsymbol{C}(n|n-1)\boldsymbol{H}^T(n)\boldsymbol{S}^{-1}(n)$$

$$\hat{x}(n|\mathfrak{m}_n) = \hat{x}(n|\mathfrak{m}_{n-1}) + \mathbf{K}(n)\vec{\alpha}(n)$$

$$\boldsymbol{C}(n|n) = [\boldsymbol{I} - \boldsymbol{K}(n)\boldsymbol{H}(n)]\boldsymbol{C}(n|n-1)$$

Gain Matrix Updater - Score selection

```
std::tuple<double, std::error_code> GainMatrixUpdater::visitMeasurement(
     InternalTrackState trackState, Direction direction,
     const Logger& logger) const {

  ...

    ParametersVector residual;
    residual = calibrated - H * trackState.filtered;
    ACTS_VERBOSE("Residual: " << residual.transpose());

    CovarianceMatrix m =
        ((CovarianceMatrix::Identity() - H * K) * calibratedCovariance).eval();

    chi2 = (residual.transpose() * m.inverse() * residual).value();

    ACTS_VERBOSE("Chi2: " << chi2);
...
}
```

$$\vec{r}(n) = \vec{m}(n) - \mathbf{H}(n)\hat{x}(n|\mathfrak{m}_n)$$

$$\chi^2_+ = \vec{r}^T(n)[(\mathbf{1} - \mathbf{H}(n)\mathbf{K}(n))\mathbf{V}(n)]^{-1}\vec{r}(n)$$

Gain Matrix Smoother

```
Result<void> GainMatrixSmoother::calculate(
    void* ts, void* prev_ts, const GetParameters& filtered,
    const GetCovariance& filteredCovariance, const GetParameters& smoothed,
    const GetParameters& predicted, const GetCovariance& predictedCovariance,
    const GetCovariance& smoothedCovariance, const GetCovariance& jacobian,
    const Logger& logger) const {

  //...
  // Gain smoothing matrix
  // NB: The jacobian stored in a state is the jacobian from previous
  // state to this state in forward propagation
  BoundMatrix G = filteredCovariance(ts) * jacobian(prev_ts).transpose() *
                  predictedCovariance(prev_ts).inverse();


  // Calculate the smoothed parameters
  smoothed(ts) = filtered(ts) + G * (smoothed(prev_ts) - predicted(prev_ts));


  // And the smoothed covariance
  smoothedCovariance(ts) =
      filteredCovariance(ts) +
      G * (smoothedCovariance(prev_ts) - predictedCovariance(prev_ts)) *
          G.transpose();
}
```

$$\boldsymbol{G}(n) = \boldsymbol{C}(n|n)\boldsymbol{J}(n+1)^T\boldsymbol{C}(n+1|n)^{-1}$$

$$\vec{x}_s(n) = \vec{x}(n|n) + \mathbf{G}(n)(\vec{x}_s(n+1) - \vec{x}(n+1|n))$$

$$\boldsymbol{C}_s(n) = \boldsymbol{C}(n|n) + \boldsymbol{G}(n)[\boldsymbol{C}_s(n+1) - \boldsymbol{C}(n+1|n)]\boldsymbol{G}(n)^T$$

Verbose run

Start of Finding Algorithm

- Comes right after seeding is ready
- Evaluates each seed and finds the most suitable track for it

```
13:37:43    TrackFinding   DEBUG     Invoke track finding with 1 seeds.
13:37:43    TrackFinding   VERBOSE   Path aborter limit set to 121666 (full helix = 243333, previous limit = 1.79769e+308)
13:37:43    TrackFinding   VERBOSE   Entering propagation.
13:37:43    TrackFinding   VERBOSE   No Volume | Initialization.
13:37:43    TrackFinding   VERBOSE   No Volume | Current surface set to start surface undefined
13:37:43    TrackFinding   VERBOSE   No Volume | Slow start initialization through search.
13:37:43    TrackFinding   VERBOSE   No Volume | Starting from position (-0.0046, -0.0265, -0.0307) and direction (-0.2584, 0.0448, -0.9650)
13:37:43    TrackFinding   VERBOSE   BeamPipe::Barrel | Start volume resolved.
```

- Chooses an initial position (and direction) and uses the stepper to extrapolate to outer layers

Start of Finding Algorithm

- Comes right after seeding is ready
- Evaluates each seed and finds the most suitable track for it

```
13:37:43    TrackFinding   DEBUG     Invoke track finding with 1 seeds.
13:37:43    TrackFinding   VERBOSE   Path aborter limit set to 121666 (full helix = 243333, previous limit = 1.79769e+308)
13:37:43    TrackFinding   VERBOSE   Entering propagation.
13:37:43    TrackFinding   VERBOSE   No Volume | Initialization.
13:37:43    TrackFinding   VERBOSE   No Volume | Current surface set to start surface undefined
13:37:43    TrackFinding   VERBOSE   No Volume | Slow start initialization through search.
13:37:43    TrackFinding   VERBOSE   No Volume | Starting from position (-0.0046, -0.0265, -0.0307) and direction (-0.2584, 0.0448, -0.9650)
13:37:43    TrackFinding   VERBOSE   BeamPipe::Barrel | Start volume resolved.
```

- Chooses an initial position (and direction) and uses the stepper to extrapolate to outer layers

Interaction with passive layer

```
13:37:43    TrackFinding    VERBOSE    CombinatorialKalmanFilter step
13:37:43    TrackFinding    VERBOSE    SurfaceReached aborter | Target surface not set.
13:37:43    TrackFinding    VERBOSE    Perform filter step
13:37:43    TrackFinding    VERBOSE    Detected passive surface: vol=5|lay=2
13:37:43    TrackFinding    VERBOSE    Create Material output track state #0 with mask: 11101101
13:37:43    TrackFinding    VERBOSE    Material effects on surface: vol=5|lay=2 at update stage: FullUpdate (0) are :
13:37:43    TrackFinding    VERBOSE    eLoss = 0.00113636, variancePhi = 2.89492e-08, varianceTheta = 1.99159e-09, varianceQoverP = 1.46451e-14
```

- When it finds a passive layer (no sensors) calculate the material interaction and the variance it introduces in the track trajectory

How is this used? Don’t know

Interaction with active layer: measurement selector

```
13:37:43    TrackFinding    VERBOSE    CombinatorialKalmanFilter step
13:37:43    TrackFinding    VERBOSE    SurfaceReached aborter | Target surface not set.
13:37:43    TrackFinding    VERBOSE    Perform filter step
13:37:43    TrackFinding    VERBOSE    Measurement surface vol=8|lay=2|sen=96 detected.
13:37:43    TrackFinding    VERBOSE    No material effects on surface: vol=8|lay=2|sen=96 at update stage: PreUpdate (-1)
13:37:43    TrackFinding    VERBOSE    Create temp track state with mask: 00011001
13:37:43    TrackFinding    VERBOSE    Invoked MeasurementSelector
13:37:43    TrackFinding    VERBOSE    Number of selected measurements: 1, max: 10
13:37:43    TrackFinding    VERBOSE    Create SourceLink output track state #1 with mask: 11111111
```

- No calculation of material interaction
- When a measurement surface is reached, its measurements are evaluated
- **MeasurementSelector** selects the measurement with higher chi2

Verbose run - Kalman Filter

```
13:37:43    TrackFinding   VERBOSE   Invoked GainMatrixUpdater
13:37:43    TrackFinding   VERBOSE   Predicted parameters:   0.869396   -11.4385    2.97316     2.8762 -0.0430675    281.684
13:37:43    TrackFinding   VERBOSE   Predicted covariance:
     1.33475     0.536067    0.0105179  5.55754e-05   -0.0115148    -0.556183
    0.536067      65.6485   0.00275089    -0.140076    -0.004782     -62.4018
   0.0105179   0.00275089  0.000356767  3.49034e-06 -0.000723159  -0.00289201
 5.55754e-05    -0.140076  3.49034e-06  0.000304619  -1.5081e-19     0.135172
  -0.0115148    -0.004782 -0.000723159  -1.5081e-19         0.01    0.0039436
   -0.556183     -62.4018  -0.00289201     0.135172    0.0039436      89935.8
13:37:43    TrackFinding   VERBOSE   Measurement dimension: 2
13:37:43    TrackFinding   VERBOSE   Calibrated measurement: 0.869697  -12.173
13:37:43    TrackFinding   VERBOSE   Calibrated measurement covariance:
0.000208333           0
          0 0.000208333
13:37:43    TrackFinding   VERBOSE   Measurement projector H:
1 0 0 0 0 0
0 1 0 0 0 0
13:37:43    TrackFinding   VERBOSE   Gain Matrix K:
    0.999843  1.27853e-06
 1.27853e-06     0.999997
  0.00788784 -2.25065e-05
   0.0009014  -0.00214107
 -0.00862457 -2.41675e-06
  -0.0350441    -0.950256
13:37:43    TrackFinding   VERBOSE   Filtered parameters:   0.869696    -12.173    2.97318    2.87777 -0.0430683    282.381
13:37:43    TrackFinding   VERBOSE   Filtered covariance:
 0.000208301  2.66359e-10   1.6433e-06  1.87792e-07 -1.79679e-06 -7.30085e-06
 2.66359e-10  0.000208333 -4.68885e-09 -4.46057e-07  -5.0349e-10  -0.00019797
  1.6433e-06 -4.68885e-09  0.000273866 -1.00642e-07  -0.00063244  9.06306e-05
 1.87792e-07 -4.46057e-07 -1.00642e-07  4.65731e-06  1.40787e-07   0.00206656
-1.79679e-06  -5.0349e-10  -0.00063244  1.40787e-07   0.00990068  -0.00100405
-7.30085e-06  -0.00019797  9.06306e-05   0.00206656  -0.00100405      89876.4
```

- Measurement is calibrated to account to physical features of the sensors
- Projector matrix H only uses l0, l1 features of the vector state

Verbose run - Score calculation

```
13:37:43    TrackFinding   VERBOSE   Residual:  9.86179e-07 -2.33882e-06
13:37:43    TrackFinding   VERBOSE   Chi2: 0.00824666
13:37:43    TrackFinding   VERBOSE   Creating measurement track state with tip = 1
13:37:43    TrackFinding   VERBOSE   Filtering step successful with 1 branches
13:37:43    TrackFinding   VERBOSE   Stepping state is updated with filtered parameter:
13:37:43    TrackFinding   VERBOSE   ->   0.869696    -12.173    2.97318    2.87777 -0.0430683    282.381 of track state with tip = 1
```

- Calculate residual and update the tip of the track

Smoother

```
13:37:43   TrackFinding   VERBOSE   Finalize/run smoothing for track with last measurement index = 24
13:37:43   TrackFinding   VERBOSE   Apply smoothing on 25 filtered track states.
13:37:43   TrackFinding   VERBOSE   Invoked GainMatrixSmoother on entry index: 24
13:37:43   TrackFinding   VERBOSE   Getting previous track state
13:37:43   TrackFinding   VERBOSE   Start smoothing from previous track state at index: 23
13:37:43   TrackFinding   VERBOSE   Calculate smoothing matrix:
13:37:43   TrackFinding   VERBOSE   Filtered covariance:6X6 MATRIX

13:37:43   TrackFinding   VERBOSE   Jacobian: 6X6 MATRIX
13:37:43   TrackFinding   VERBOSE   Prev. predicted covariance.inverse 6X6 MATRIX
13:37:43   TrackFinding   VERBOSE   Gain smoothing matrix G: 6X6 MATRIX
13:37:43   TrackFinding   VERBOSE   Calculate smoothed parameters:
13:37:43   TrackFinding   VERBOSE   Filtered parameters:   689.019   3.02277   3.08182   2.87751 -0.0740822   2802.67
13:37:43   TrackFinding   VERBOSE   Prev. smoothed parameters:   -35.1364   -33.3258   3.10037   2.87752 -0.0743785   3211.36
13:37:43   TrackFinding   VERBOSE   Prev. predicted parameters:   -35.1833   -33.3035   3.09997   2.87751 -0.0740894   3211.36
13:37:43   TrackFinding   VERBOSE   Smoothed parameters are:   689.002   3.02278   3.08195   2.87753 -0.0743714   2802.67
13:37:43   TrackFinding   VERBOSE   Calculate smoothed covariance: 6X6 MATRIX
13:37:43   TrackFinding   VERBOSE   Prev. smoothed covariance: 6X6 MATRIX

13:37:43   TrackFinding   VERBOSE   Smoothed covariance is: 6X6 MATRIX
```

Open Questions

How …

- the first point defined?
- the first prediction is established?
- is energy (q/p) and time (t) parameters transported?

Things I still don't understand that well

- Stepper covariance transport
- Using C(n) matrix instead of P(n)
- Calculation of the k coefficients in the Stepper
- Where the sensor reading is used (outside clustering)

Why …

- material interaction with passive layers is accounted for
- the sensor reading is not used

Next steps

Explore the ACTS track reconstruction framework (on going)

- **Next ?:** Continue investigating the CKF until a full understanding
- Study the implementation of the GSF in the core library
- Get the ITk and HGTD geometry to work in the ACTS

Follow HGTD ACTS integration campaign

- Meeting scheduled!

Theoretical Study

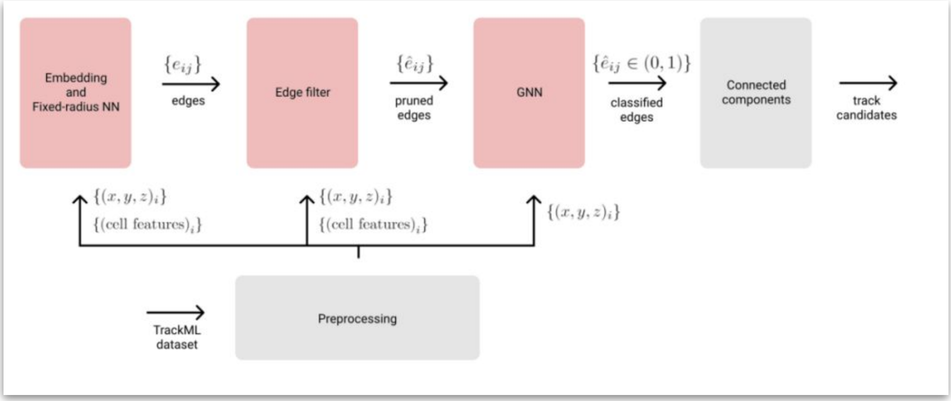
- H. Kolanosky, Particle Detectors (2020)
  - Finished Chapters 1-5
  - Next: Chapters 8-9
- Read papers of systematic review

Backup

HGTD presentation sketch

ATLAS 4D track reconstruction at the HL-LHC

- Study path reconstruction methods adapted to ATLAS in the HL-LHC
- **Using the HGTD timing information to perform 4D reconstruction!**
- Will explore Machine Learning and Deep Learning approaches as they showed promising results in previous studies
- Important to propose a method that is efficient and optimized for parallel processing (GPU)



Stages of the TrackML track formation inference pipeline (Ju X et. al., 2021)

- Aware of many ongoing studies in the area
- Most of the existing proposals:
  - Are tested in the ODD (Open Data Detector)
  - Don't integrate HGTD timing and ToT (Time over Threshold) information

HGTD and ACTS familiarity

HGTD and ATLAS

- I’ve read the performance section of the HGTD TDR
- Basic knowledge of LGAD sensors
  - Following Prof. Marco’s sensor development and characterization
- Familiarity with ATLAS DAQ and trigger architecture
  - Worked with the LAr operations team for a while

ACTS and reconstruction methods

- Took lecture courses in signal processing and adaptive filters (including Kalman Filters)
  - My second advisor is a specialist
- Started using ACTS -> Next slides!

Understanding the CKF

2.a - Defining a state space

- We can define the measures we observe as a function of the true position and measurement errors
- A simple measurement equation would be:

$$\vec{m}(n) = \mathbf{H}(n)\vec{x}(n) + \vec{\epsilon}(n)$$

n	layer (surface) index
$\vec{m}(n)$	measure vector (x,y,z)
$\mathbf{H}(n)$	projection matrix (x to m)
$\vec{x}(n)$	state vector ⇦ **Value to be estimated**
$\vec{\epsilon}(n)$	measurement error, white gaussian noise

- As we know the system dynamics, we can also define a system equation

$$\vec{x}(n) = \mathbf{F}(n-1)\vec{x}(n-1) + \vec{\omega}(n-1)$$

$\mathbf{F}(n-1)$	transport state vector from (n-1) to (n)
	*extrapolation achievable by numeric integration
$\vec{\omega}(n)$	system error, white gaussian noise
	is independent of the measurement error

- These two equations define our state space

2.b - Innovation process and estimative update

- If we have a prior estimative of the state vector (before observing the actual measurement) is possible to define a metric that measures the information gain that the new measurement offers
- The innovation is achievable with the following equations:

$$\vec{\alpha}(n) = \vec{m}(n) - \hat{m}(n)$$
$$\vec{\alpha}(n) = \vec{m}(n) - \mathbf{H}(n)\hat{x}(n|\mathfrak{m}_{n-1})$$

$\hat{m}(n) = \mathbb{E}[\vec{m}(n)] = \mathbf{H}(n)\vec{x}(n)$

$\hat{x}(n|\mathfrak{m}_{n-1})$ estimative of state vector given prior measures

- The innovation can be used to adjust the prior estimative:

$$\hat{x}(n|\mathfrak{m}_n) = \hat{x}(n|\mathfrak{m}_{n-1})) + \mathbf{K}(n)\vec{\alpha}(n)$$

- Where $\mathbf{K}(n)$ is the Kalman gain, which is chosen to minimize the mean-square value of the estimation error

$$\varepsilon(n|n) = \vec{x}(n) - \hat{x}(n|\mathfrak{m}_n)$$

$$\mathbb{J} = \mathbb{E}\{||\varepsilon(n|n)||^2\}$$

Finding the Kalman gain

- Defining the following correlation matrixes

$$\mathbf{P}(n|n) = \mathbb{E}\{\varepsilon(n|n)\varepsilon^T(n|n)\} \qquad \mathbf{S}(n) = \mathbb{E}\{\vec{\alpha}(n)\vec{\alpha}^T(n)\}$$

- We can express our error metric in function of P

$$\mathbb{J} = \mathbb{E}\{||\varepsilon(n|n)||^2\} = \mathrm{tr}[\mathbf{P}(n|n)]$$

- Then we just need to find the argument K that minimizes the metric

$$\mathbb{J}(\mathbf{K}(n)) = \mathrm{tr}[\mathbf{P}(n|n{-}1)] - 2\mathrm{tr}[\mathbf{K}(n)\mathbf{H}(n)\mathbf{P}(n|n{-}1)] + \mathrm{tr}[\mathbf{K}(n)\mathbf{S}(n)\mathbf{K}^T(n)]$$

$$\mathbf{K}^o(n) = \mathbf{P}(n|n-1)\mathbf{H}^T(n)\mathbf{S}^{-1}(n)$$

Kalman Filter Summary

- Iteration between **prior estimative** and **filtered estimative (posteriori)**

$$\vec{x}(n|\mathfrak{m}_n) = \vec{x}(n|\mathfrak{m}_{n-1}) + \mathbf{K}(n)\vec{\alpha}(n)$$

$$\vec{x}(n+1|\mathfrak{m}_n) = \mathbf{F}(n)\vec{x}(n|\mathfrak{m}_n)$$

- It's also necessary to iterate over error estimative matrixes in order to calculate the Kalman gain

$$\boldsymbol{P}(n|n) = [\boldsymbol{I} - \boldsymbol{K}(n)\boldsymbol{H}(n)]\boldsymbol{P}(n|n-1)$$

$$\boldsymbol{P}(n+1|n) = \boldsymbol{F}(n)\boldsymbol{P}(n|n)\boldsymbol{F}^T(n) + \mathbf{V}_x$$

$$\mathbf{K}^o(n) = \mathbf{P}(n|n-1)\mathbf{H}^T(n)\mathbf{S}^{-1}(n)$$

- After all measures are available, it is also possible to smooth the estimates.

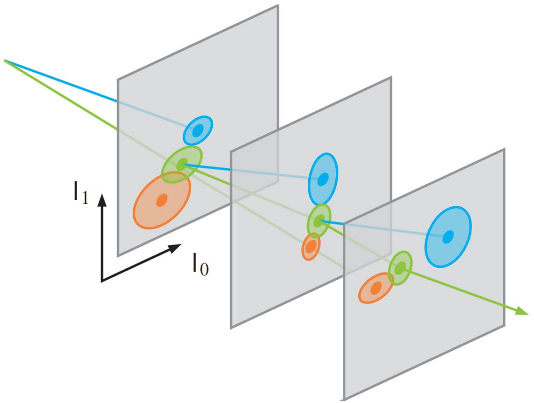


Figure 2.1: Illustration of KF estimative iteration. Measurement represented in orange, (prior) estimative in blue and filtered (posteriori) estimative in green.

Numerical Integration for Track extrapolation

- Numerical integration is done using the fourth order *Range-Kutta-Nystrom (RKN)* method

- The RKN solves a problem that can described as:

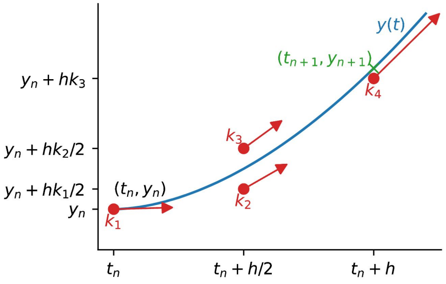
$$\frac{dy}{dt} = f(t, y), \qquad y(t_0) = y_0,$$

- our function can be defined as

$$\frac{d^2\vec{r}}{ds^2} = \frac{q}{p}\left(\frac{d\vec{r}}{ds} \times \vec{B}(\vec{r})\right) = f(s, \vec{r}, \vec{T}), \qquad \vec{T} \equiv \frac{d\vec{r}}{ds},$$

- The function *f* is evaluated at four points:

$$\begin{aligned} k_1 &= f(t_n, y_n) \\ k_2 &= f\left(t_n + \frac{h}{2}, y_n + h\frac{k_1}{2}\right) \\ k_3 &= f\left(t_n + \frac{h}{2}, y_n + h\frac{k_2}{2}\right) \\ k_4 &= f\left(t_n + h, y_n + hk_3\right). \end{aligned}$$



- Finally, these points are used to generate an estimate of the next state

$$\vec{T}_{n+1} = \vec{T}_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

$$\vec{r}_{n+1} = \vec{r}_n + h\vec{T}_n + \frac{h^2}{6}(k_1 + k_2 + k_3).$$

ACTS usage

ACTS Setup

- Using a machine running CVMFS (CernVM File System) all dependencies can be easily satisfied via a LCG release. For this case, a setup file is provided.
  - As SAMPA (IFUSP cluster) runs CVMFS, we will use it

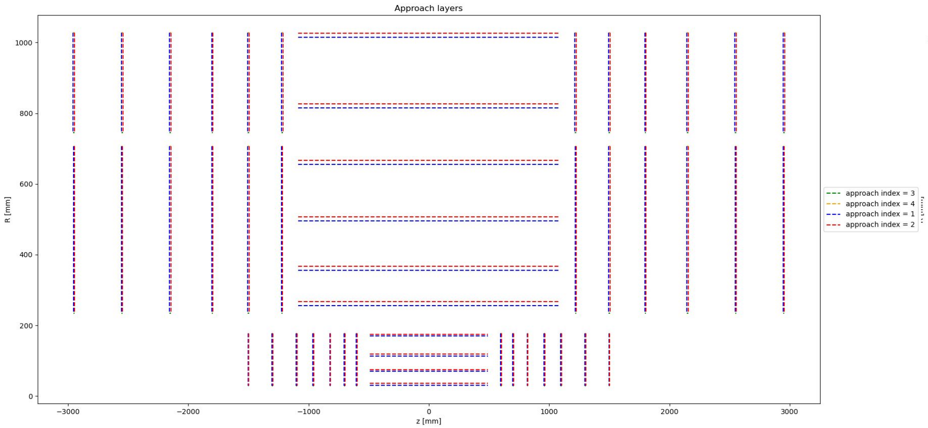
```
git clone https://github.com/acts-project/acts <source>
cd <source>
source CI/setup_cvmfs_lcg.sh
cmake -B build -S . -DACTS_BUILD_FATRAS=on -DACTS_BUILD_EXAMPLES_PYTHON_BINDINGS=ON
cmake --build build
```

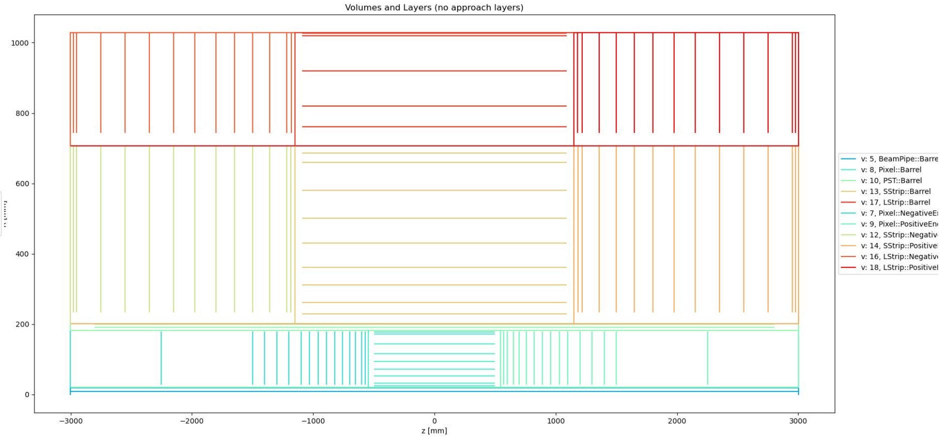
- To use Python bindings, it is also necessary to setup a Python env

```
python -m venv acts
source acts/bin/activate
source $BUILD_DIR/python/setup.sh
```

Geometry visualization

- ACTS is independent of detector geometry, so the user can choose what geometry to use
- Open Data Detector (ODD) that provides a generic tracking detector is used as base in the ACTS if no geometry is provided
  - Geometry file can be generated with the script <source>/Examples/Scripts/Python/geometry.py
  - And printed with the script <source>/Examples/Scripts/MaterialMapping/GeometryVisualisationAndMaterialHandling.py





- **Next steps:** get ITk + HGTD geometry files
  - They're not fully implemented but some examples already use this geometry

Particle gun setting

```python
s = s or acts.examples.Sequencer(
    events=100, numThreads=-1, logLevel=acts.logging.INFO
)

for d in decorators:
    s.addContextDecorator(d)

rnd = acts.examples.RandomNumbers(seed=42)
outputDir = Path(outputDir)

if inputParticlePath is None:
    addParticleGun(
        s,
        ParticleConfig(num=1,
                       pdg=acts.PdgParticle.eMuon,
                       randomizeCharge=True
                      ),
        EtaConfig(-3.0, 3.0, uniform=True),
        MomentumConfig(1.0 * u.GeV,
                       100.0 * u.GeV,
                       transverse=True
                      ),
        PhiConfig(0.0, 360.0 * u.degree),
        vtxGen=acts.examples.GaussianVertexGenerator(
            mean=acts.Vector4(0, 0, 0, 0),
            stddev=acts.Vector4(0, 0, 0, 0),
        ),
        multiplicity=1,
        rnd=rnd,
    )
```

- The sequencer defines the processing chain, so we plug steps into it. The first step being the particle gun

particle_id	4503599644147712
particle_type	13
process	0
vx	0
vy	0
vz	0
vt	0
px	375.925.779
py	560.662.365
pz	299.960.766
m	105.658.367
q	-1

Example of particle gun output

FATRAS Propagation

```
addFatras(
    s,
    trackingGeometry,
    field,
    rnd=rnd,
    enableInteractions=True,
)
```

- Produce the hits on our detector
- FATRAS uses parametrized equations that describe the interaction of particles with matter (detector layers)
  - Bethe-Bloch and Bethe-Heitler
  - [Description in the documentation]
- Using Geant4 is also possible

particle_id	geometry_id	tx	ty	tz	tt	tpx	tpy	tpz	te	deltapx	deltapy	deltapz	deltae	index
4503599644147	5764608897424	17.6435509	26.3971519	141.092026	0.482404649	3.74247098	5.61727953	29.9951477	30.7454052	-0.00012434988	-0.00011105436	-0.00017800545	-0.00020908787	0
4503599644147	5764610271814	39.398716	59.1917572	315.947205	1.08025253	3.72160625	5.63174248	29.9916401	30.7420959	-0.0015875214	-0.00108358543	0.000240411115	-0.00015608617	1
4503599644147	5764610271814	40.486618	60.8387337	324.719391	1.11024511	3.7190311	5.63131142	29.9918804	30.7419395	0.00132368144	0.00286062085	-0.00088070239	-0.00017489859	2
4503599644147	6485184837803	74.7456436	113.032188	602	2.05831456	3.68934417	5.6548562	29.9871826	30.7381039	7.51150219e-05	-0.00016811005	-0.00011313620	-0.00013228319	3
4503599644147	6485186212192	87.0301437	131.90155	702	2.40023136	3.67800283	5.66169071	29.9866791	30.7375088	0.000437328505	-0.00033966929	-6.51633745e-05	-7.38019371e-05	4
4503599644147	1008806453969	149.506027	229.198639	1215.5	4.15598202	3.61827326	5.70087671	29.9836063	30.7346668	0.000204591735	0.00011170045	-0.00010522620	-5.78490362e-05	5
4503599644147	1008806591408	183.202545	282.533051	1495.5	5.11338043	3.59212995	5.72083426	29.9808102	30.7325802	-0.00044944352	0.000355006661	-9.88422253e-05	-8.28674238e-05	6
4503599644147	1008806728847	220.028687	341.603912	1804.5	6.16992998	3.5551486	5.74233389	29.9792233	30.7307415	-0.00139961659	0.000372228649	2.14714364e-05	-7.13826084e-05	7
4503599644147	1008806866286	261.282684	408.792969	2154.5	7.36666679	3.51257873	5.76569653	29.9784126	30.7294292	-4.11071269e-05	0.000126262108	-0.00020031817	-0.00017643056	8
4503599644147	1008807003725	306.801514	484.173828	2545.5	8.70357704	3.46734047	5.79311514	29.9782124	30.7292538	0.000675019168	3.89658308e-05	-0.00016233704	-7.48498787e-05	9
4503599644147	1008807141164	352.749023	561.686523	2945.5	10.0712767	3.4201479	5.82276011	29.9769821	30.7283688	1.92528096e-05	0.00021961586	-0.00017930175	-0.00013115815	10

Digitization

```python
addDigitization(
    s,
    trackingGeometry,
    field,
    addFatras(
    s,
    trackingGeometry,
    field,
    rnd=rnd,
    enableInteractions=True,
)
```

- Simulate the measure by the pixels of the detector layers
- Clustering already included (?)
    - Assuming this as we have var_local0 and var_local1

measurement_id	geometry_id	local_key	local0	local1	phi	theta	time	var_local0	var_local1	var_phi	var_theta	var_time
0	5764608897424	□	-12.594.698	-339.011.307	0	0	0	208.333.338	208.333.338	0	0	0
1	5764610271814	□	715.210.676	918.191.373	0	0	0	208.333.338	208.333.338	0	0	0
2	5764610271814	□	-709.059.334	972.053.432	0	0	0	208.333.338	208.333.338	0	0	0
3	6485184837803	□	-221.331.811	-451.492.071	0	0	0	208.333.338	208.333.338	0	0	0
4	6485186212192	□	-273.950.982	180.044.041	0	0	0	208.333.338	208.333.338	0	0	0
5	1008806453969	□	-101.005.602	-445.343.361	0	0	0	533.333.339	119.999.997	0	0	0
6	1008806591408	□	-223.774.886	18.994.276	0	0	0	533.333.339	119.999.997	0	0	0
7	1008806728847	□	-182.034.855	-642.108.154	0	0	0	533.333.339	119.999.997	0	0	0
8	1008806866286	□	-234.517.422	141.149.931	0	0	0	533.333.339	119.999.997	0	0	0
9	1008807003725	□	-640.522.623	-489.026.489	0	0	0	533.333.339	119.999.997	0	0	0
10	1008807141164	□	-100.600.576	40.876.564	0	0	0	533.333.339	119.999.997	0	0	0

Seeding

```
addSeeding(
    s,
    trackingGeometry,
    field,
    rnd=rnd,
    inputParticles="particles_input",
    seedingAlgorithm=SeedingAlgorithm.TruthSmeared,
    particleHypothesis=acts.ParticleHypothesis.muon,
    truthSeedRanges=TruthSeedRanges(
        pt=(1 * u.GeV, None),
        nHits=(7, None),
    ),
)
```

- Implement the Seeding step
- Highly customizable
- Return the tracks to be evaluated by the fitter and the finder

Could not turn on debug options to see the generated data :(

Fitting and Finding

```
addKalmanTracks(
    s,
    trackingGeometry,
    field,
    directNavigation,
    reverseFilteringMomThreshold,
)

s.addAlgorithm(
    acts.examples.TrackSelectorAlgorithm(
        level=acts.logging.INFO,
        inputTracks="tracks",
        outputTracks="selected-tracks",
        selectorConfig=acts.TrackSelector.Config(
            minMeasurements=7,
        ),
    )
)
```

- In this example the reconstruction is done in two steps but can be merged if the function *addCKFTracks()* is used
- Results in the next slides
  - Output ROOT files that compare truth tracks with reconstructed ones