# ATLAS 4D track reconstruction at the HL-LHC

Rodrigo Estevam de Paula

Advisors:

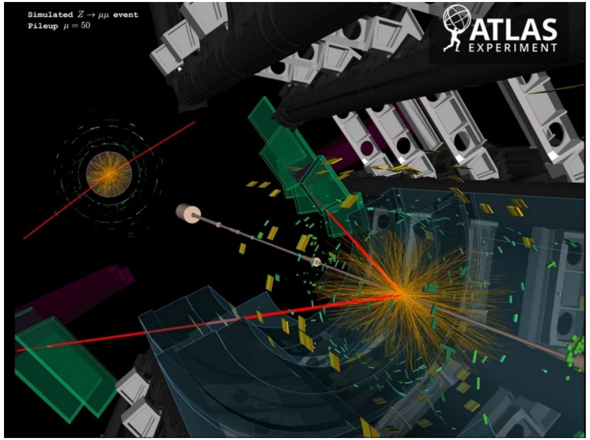
Marco Aurelio Lisboa Leite

Vitor Heloiz do Nascimento

February 22, 2024

# Challenges of track reconstruction at the HL-LHC

- For the HL-LHC the expected pileup is up to $< \mu > \approx 200$ simultaneous collisions. A significant increase from the $< \mu > \approx 65$ of Run 3.
- The consequent increase in occupancy of the tracking detector imposes many challenges in the reconstruction:
    - Harder to differentiate Hard Scatter from Pileup
    - Increase in information data to be processed in the same time frame



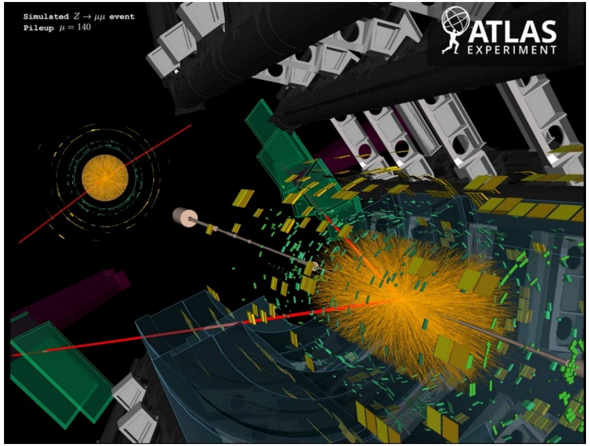




Figure 1: Simulation of a $Z \to \mu\mu$ event in the ATLAS experiment with $< \mu > = 50$ (left) and $< \mu > = 140$ (right)

# ATLAS preparation for tracking in the HL-LHC

- The Inner Detector (ID) will be substituted to a more segmented detector, the Inner Tracker (ITk)

- An additional detector, called HGTD, will be installed in the endcap region to complement the tracking in the region where the ITk resolution is insufficient ($2.4 <|\eta|< 4.0$)
  - Tracks of the ITk will be associated with hits in HGTD
  - HGTD also provides time information (not present in the ITk) that can help with pileup rejection

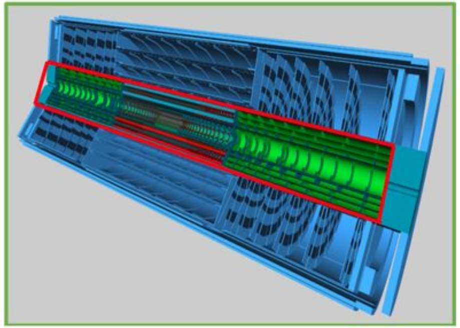
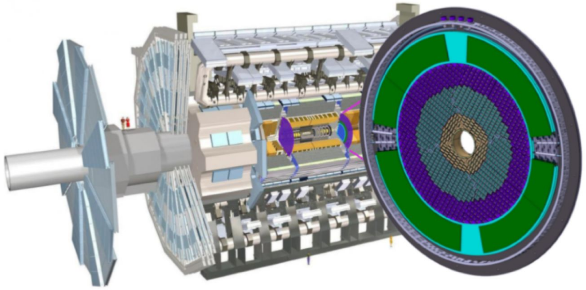



Figure 2: ITk layout (left) and HGTD position within the ATLAS experiment (right)

- **New track reconstruction methods that are adapted to the new detectors and meet the requirements imposed by the HL-LHC scenario need to be devised!**

# Outline

## Existing techniques

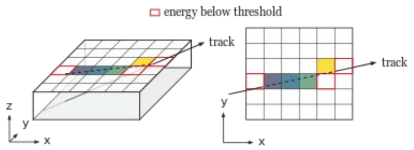
1. ATLAS tracking particle process
2. Combinational Kalman Filters applied for track fitting and finding

Understanding the existing scenario gives us a baseline performance to be expected of new methods

## Proposal for HL-LHC

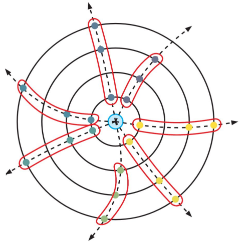
3. ACTS simulator first steps
4. New methods proposal
5. Next steps

# Track reconstruction processing chain



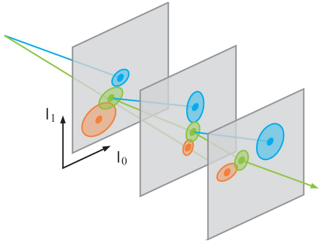

Clustering

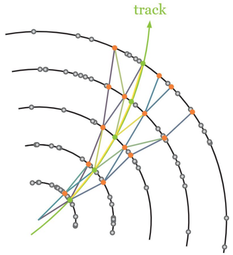
Group together sensor activations that belong to the same deposition



Seeding

Evaluate triples of deposition in the inner layers to filter the best candidates for track seeds




Track fitting and finding

# Combinational Kalman Filters

# 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)<br>*extrapolation achievable by numeric integration |
| $\vec{\omega}(n)$ | system error, white gaussian noise<br>is independent of the measurement error |

- These two equations define our state space

# 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

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

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

# Filtering Estimatives

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

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

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

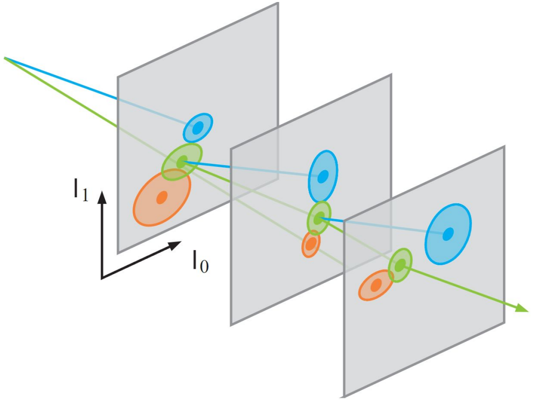



Figure 4: Illustration of KF estimative iteration. Measurement represented in orange, (prior) estimative in blue and filtered (posteriori) estimative in green [4].

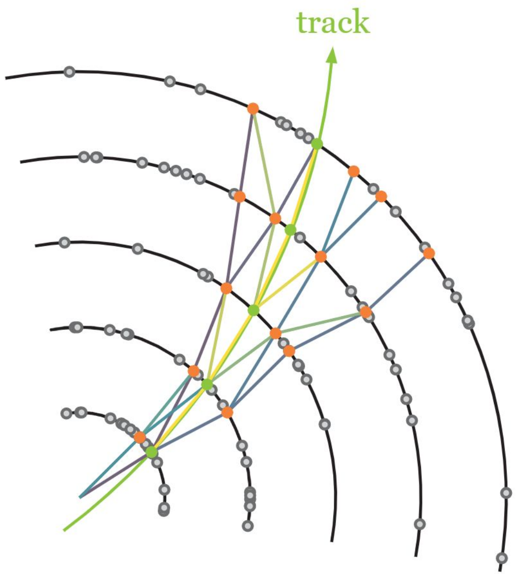
# Track scoring

- We can define a residual between the posteriori estimate and the measure

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

- This value contributes to a quality factor of the reconstructed track

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




- The algorithm iterates over all possible tracks and uses the global quality parameter $\chi^2$ (also depends on other track attributes*) to filter the best estimate tracks

# Combinational Kalman Filter Drawbacks and Limitations

- The main drawback of the CKF is the execution time that will not be acceptable in a scenario of high luminosity
    - Execution time is $O(N^2)$, being N the number of points to be fitted
    - As it is implemented today, the iterative adjustments are optimized for CPU processing

- As the Inner Detector (ID) has no timing information, and therefore the reconstruction is done in 3 dimensions
    - With the timing information provided by the HGTD the reconstruction can be made in 4 dimensions (geometrical position + time)

# ACTS first steps

# ACTS - A Common Tracking Software

- “ACTS is an experiment-independent toolkit for (charged) particle track reconstruction in (high energy) physics experiments implemented in modern C++”[4]
- Originated from Athena (ATLAS simulation framework) as a standalone version of its tracking reconstruction
- Key features:
  - A tracking geometry description, which can be constructed manually or from TGeo and DD4hep input.
  - Simple event data model.
  - Implementations of common algorithms
    - for track propagation and fitting.
    - basic seed finding.
    - vertexing.
- [Documentation website]

Runs smoothly in the SAMPA cluster 😎

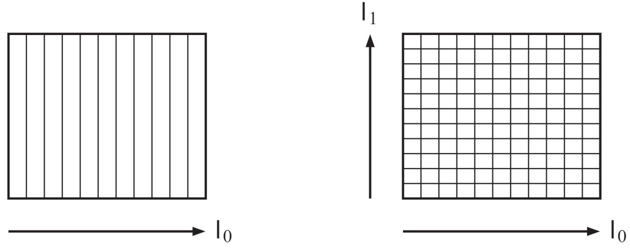
# Simulating events with particle guns

- ACTS offers a series of examples of Python bindings that can be used to simulate basic scenarios
  - The use of these bindings in “production” is encouraged
- We are going to simulate the reconstruction done by the Combinational Kalman Filter
- The Setup will be
  - ODD detector structure
  - Particle gun of 100 muons distributed uniformly between $\eta$ -3 and 3
  - Using “standard” seeding algorithms
  - Using CKF to reconstruct the tracks

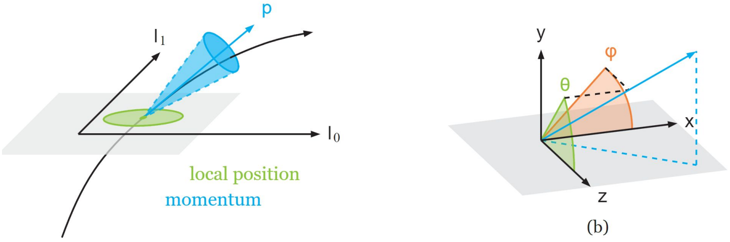
# State vector and analysis

- The simulation outputs ROOT files that can be analysed for performance evaluation.
    - We will use ACTS analysis application to generate the performance plots
    - Can do our analysis in the future
- As we know the real particle paths is possible to extract residual metrics
- The state vector is defined as:

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

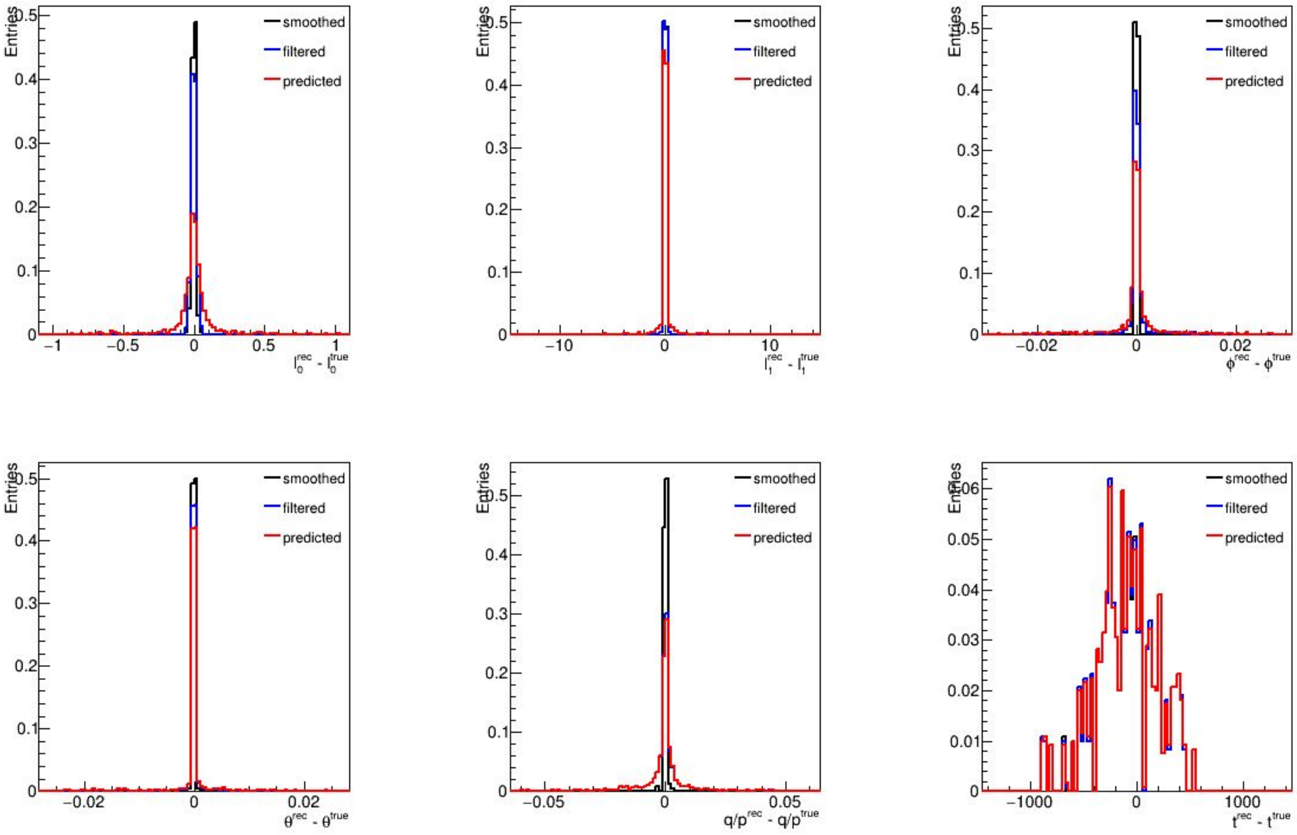



(a) strip (b) pixel

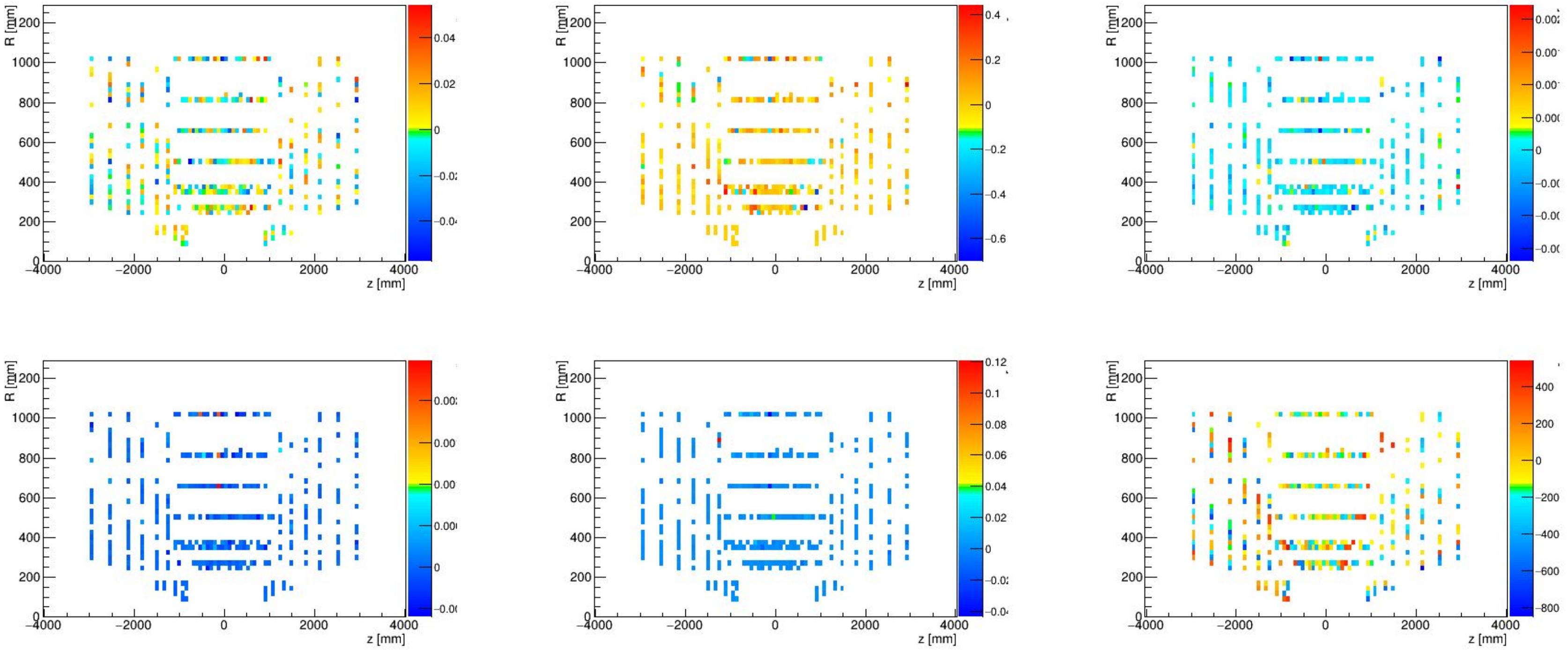



(a) (b)

# Kalman Filter performance (muons) - Residual plots



# Kalman Filter performance (muons) - Regional residual plots






More plots in the backup

# New methods proposal

# Proposal of new reconstruction techniques

## Adaptative Filter techniques

### Track fitting

- Substitute Kalman for RLS algorithms (less accurate but faster)
  - Suggested on Haykin’s book [1]
  - Combination of Adaptative Filters

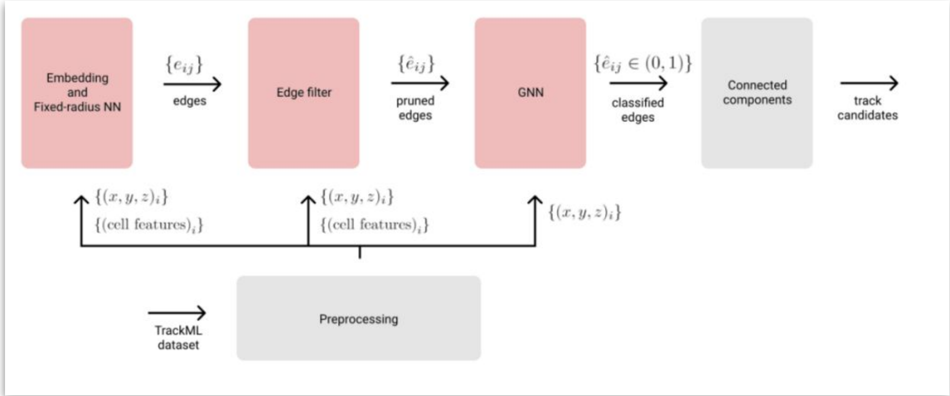
### Track finding

- Include HGTD hits in the cumulative score
  - Use additional time information in the decision process

Conservative expansion of the expected baseline performance

## Machine Learning Techniques

- Study the usage of Graph Neural Networks (GNNs) for track reconstruction
  - First proposed in the TrackML challenge
  - Study how to adapt it to 4D reconstruction




Pipeline of the exa.trk algorithm proposed at the TrakML challenge. Red stages represent stages where the model can be trained [6].

# Next steps

## Explore the ACTS track reconstruction framework (on going)

- Simulation of collision events
  - Done, but need validation
- Study the implementation of the CKF and GSF in the core library
- Get the ITk and HGTD geometry to work in the ACTS

## Follow HGTD ACTS integration campaign

- Contribute with the Simulation and Performance team

## Study more about Machine Learning based reconstruction

- GNNs
- Physics Informed Machine Learning

# Thank you for your attention!

Questions?

# References


[1] Simon Haykin. *Adaptive filter theory*. Prentice Hall, Upper Saddle River, NJ, 4th edition, 2002.

[2] Paul Gessinger-Befurt. *Development and improvement of track reconstruction software and search for disappearing tracks with the ATLAS experiment*, 2021. Presented 30 Apr 2021.

[3] Maria D. Miranda. *PTC5890: Adaptive Filters*. Graduation course at Poli-USP

[4] ATLAS Collaboration. *ACTS documentation.*
Available at: https://acts.readthedocs.io/en/latest/index.html

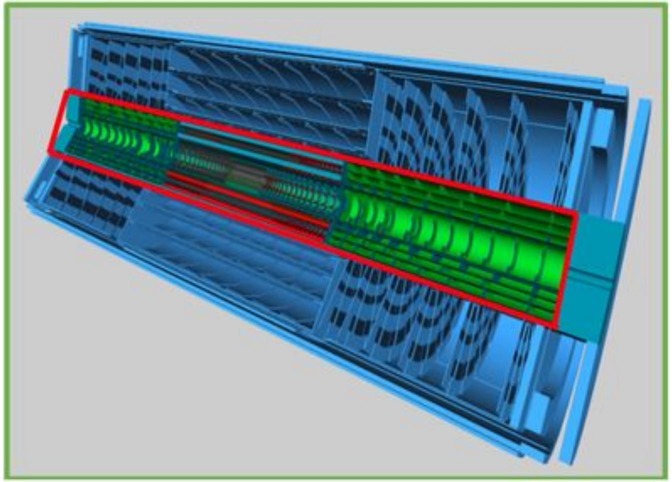
[5] Kolanoski, Hermann, and Norbert Wermes, *Particle Detectors: Fundamentals and Applications* (Oxford, 2020; online edn, Oxford Academic, 17 Sept. 2020), https://doi.org/10.1093/oso/9780198858362.001.0001 , accessed 19 Feb. 2024.

[6] JU, X. et al. *Performance of a geometric deep learning pipeline for HL-LHC particle tracking*. The European Physical Journal C, v. 81, n. 10, p. 876, out. 2021. ISSN 1434-6052. DOI: 10.1140/ epjc/s10052- 021- 09675- 8. Available at https://doi.org/10.1140/epjc/s10052-021-09675-8 .

# Backup

# Detector layout and working principle

- The track reconstruction is realized by a highly segmented detector formed by semiconductor sensors
- Electromagnetic particles ionizes the sensors and the charges are guided through an electrode and read by a readout chain



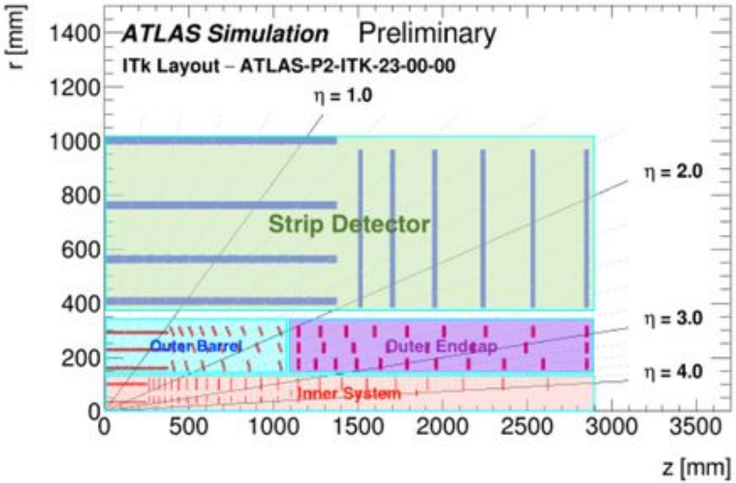



Figure 3: ATLAS Inner Tracker (ITk) layout

# Finding the Kalman gain

- Defining the following correlation matrixes

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

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

- 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

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

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

# ACTS w/ Particle Guns

# 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

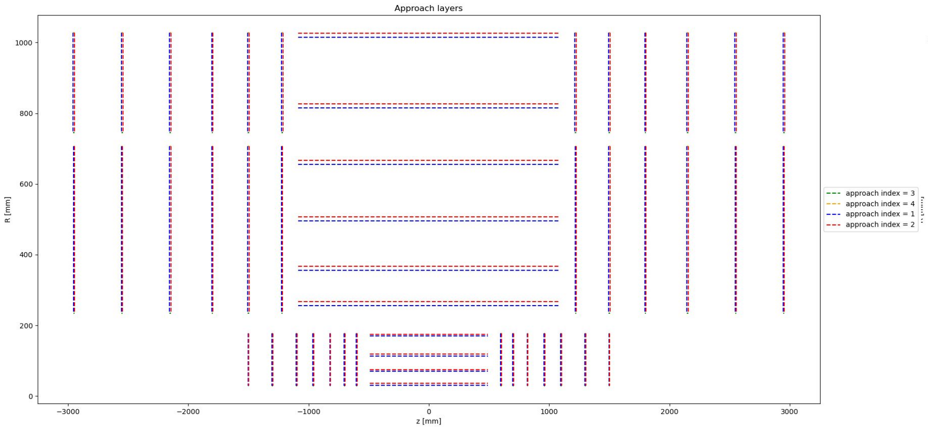
```
git clone https://github.com/acts-project/acts <source>
cd <source>
source CI/setup_cvmfs_lcg.sh
cmake -B build -S . -DACTS_BUILD_FATRAS=on -DACTS_BUILD_EXAMPLES_PYTHON_BINDINGS=ON
cmake --build build
```

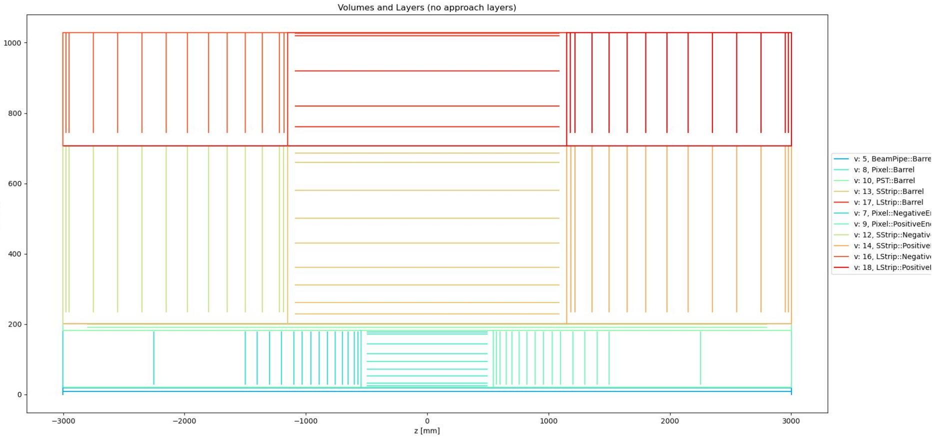
- To use Python bindings, it is also necessary to setup a Python env

```
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

```
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

```python
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 | 0.00043732850 | -0.00033966929 | -6.51633745e-0 | -7.38019371e-0 | 4 |
| 4503599644147 | 1008806453969 | 149.506027 | 229.198639 | 1215.5 | 4.15598202 | 3.61827326 | 5.70087671 | 29.9836063 | 30.7346668 | 0.00020459173 | 0.00011170045 | -0.00010522620 | -5.78490362e-0 | 5 |
| 4503599644147 | 1008806591408 | 183.202545 | 282.533051 | 1495.5 | 5.11338043 | 3.59212995 | 5.72083426 | 29.9808102 | 30.7325802 | -0.00044944352 | 0.00035500666 | -9.88422253e-0 | -8.28674238e-0 | 6 |
| 4503599644147 | 1008806728847 | 220.028687 | 341.603912 | 1804.5 | 6.16992998 | 3.5551486 | 5.74233389 | 29.9792233 | 30.7307415 | -0.00139961659 | 0.00037222864 | 2.14714364e-05 | -7.13826084e-0 | 7 |
| 4503599644147 | 1008806866286 | 261.282684 | 408.792969 | 2154.5 | 7.36666679 | 3.51257873 | 5.76569653 | 29.9784126 | 30.7294292 | -4.11071269e-0 | 0.00012626210 | -0.00020031817 | -0.00017643056 | 8 |
| 4503599644147 | 1008807003725 | 306.801514 | 484.173828 | 2545.5 | 8.70357704 | 3.46734047 | 5.79311514 | 29.9782124 | 30.7292538 | 0.00067501916 | 3.89658308e-05 | -0.00016233704 | -7.48498787e-0 | 9 |
| 4503599644147 | 100880714116 | 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

```
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

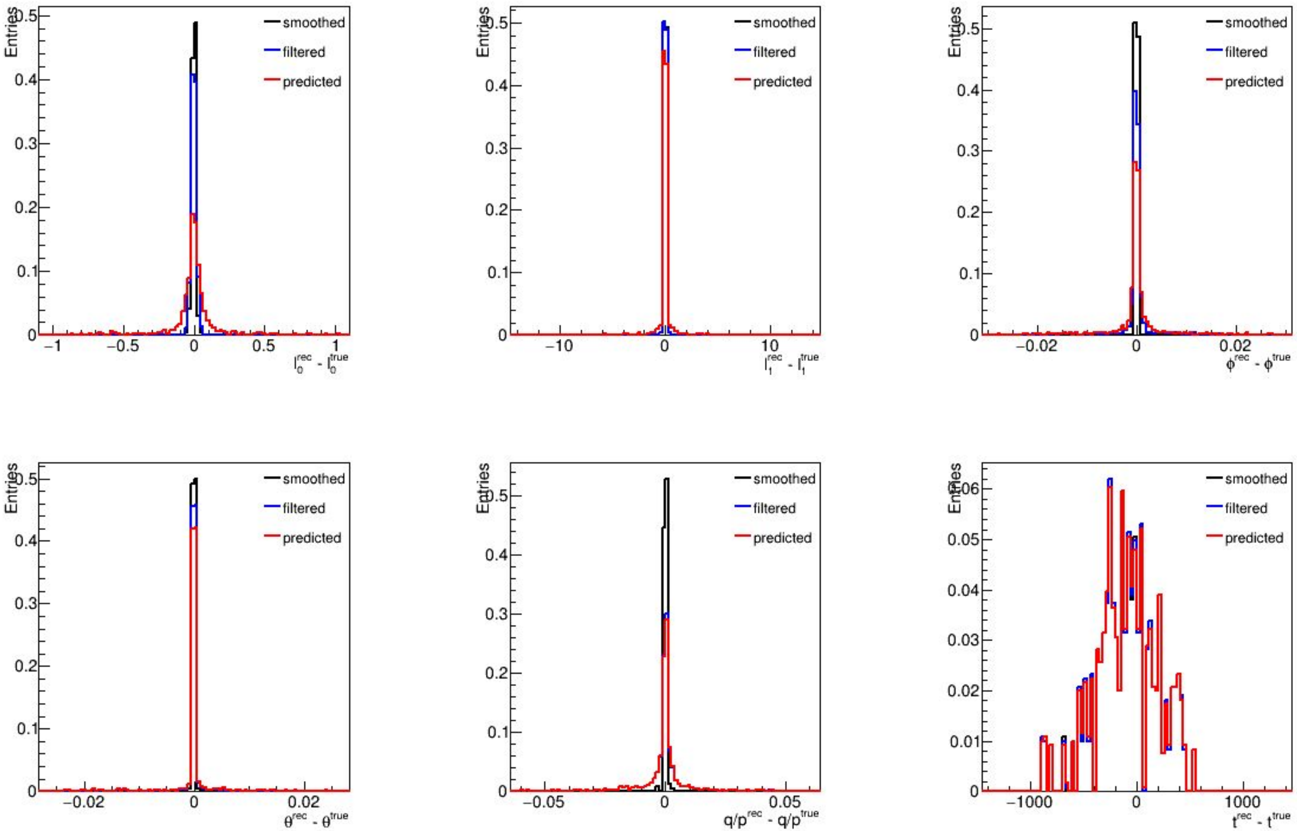
Could not turn on debug options to see the generated data :(

# Fitting and Finding

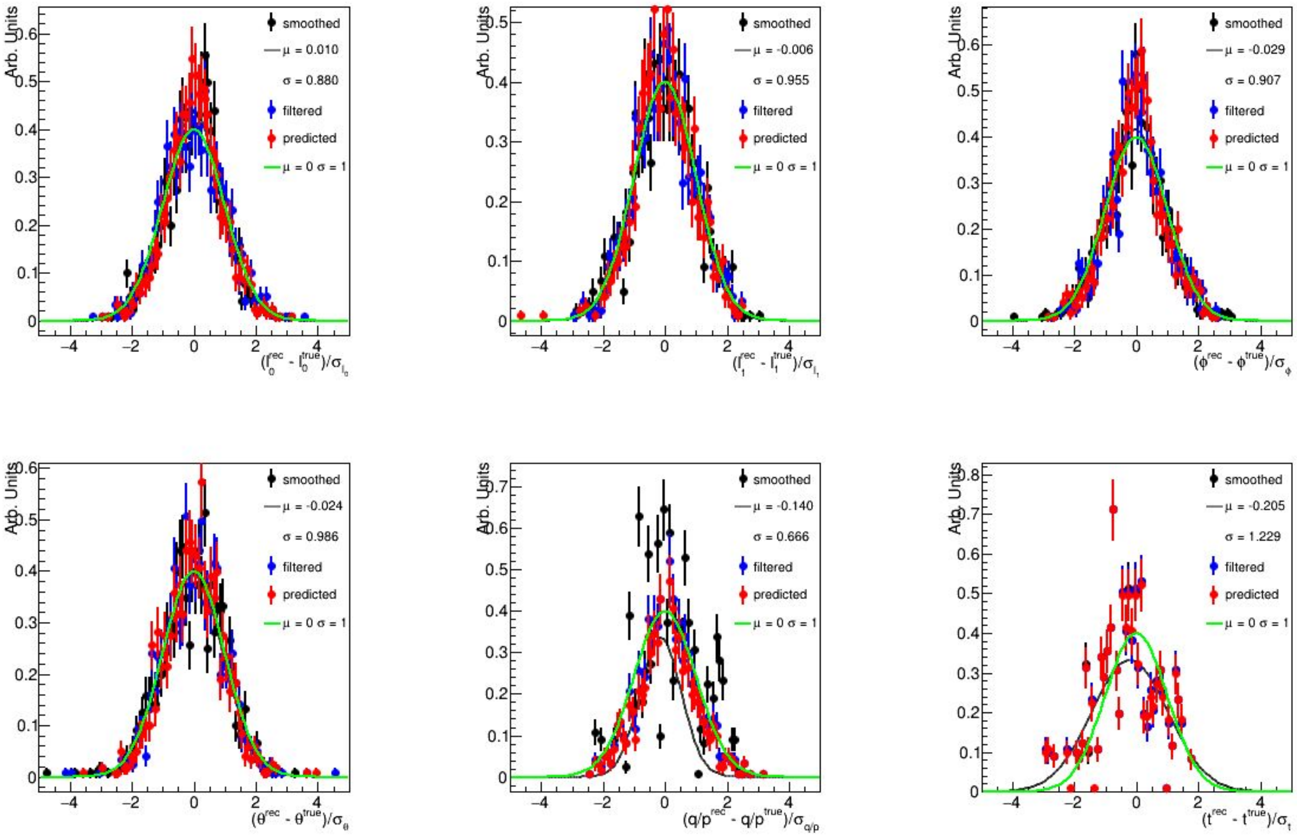
```
addKalmanTracks(
    s,
    trackingGeometry,
    field,
    directNavigation,
    reverseFilteringMomThreshold,
)

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

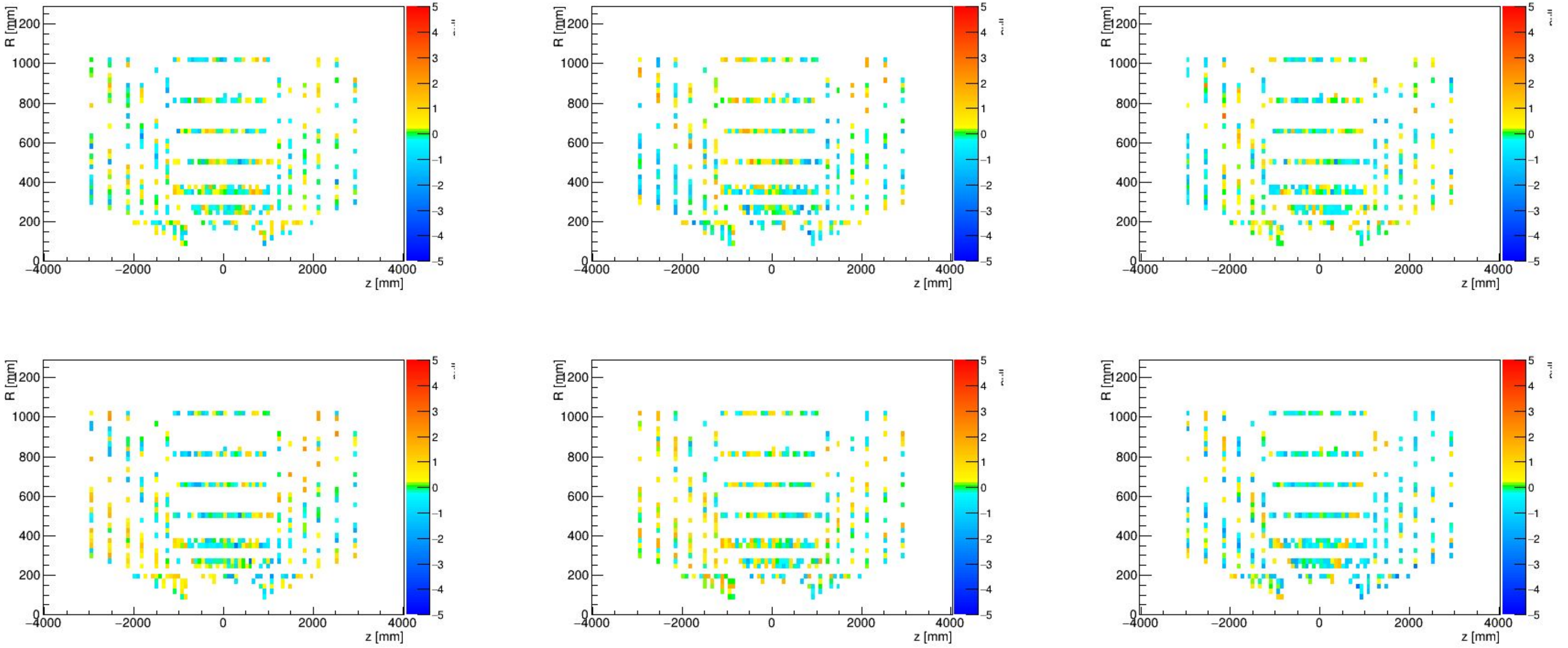
# Kalman Filter performance (muons) - Residual plots



# Kalman Filter performance (muons) - Pull plots

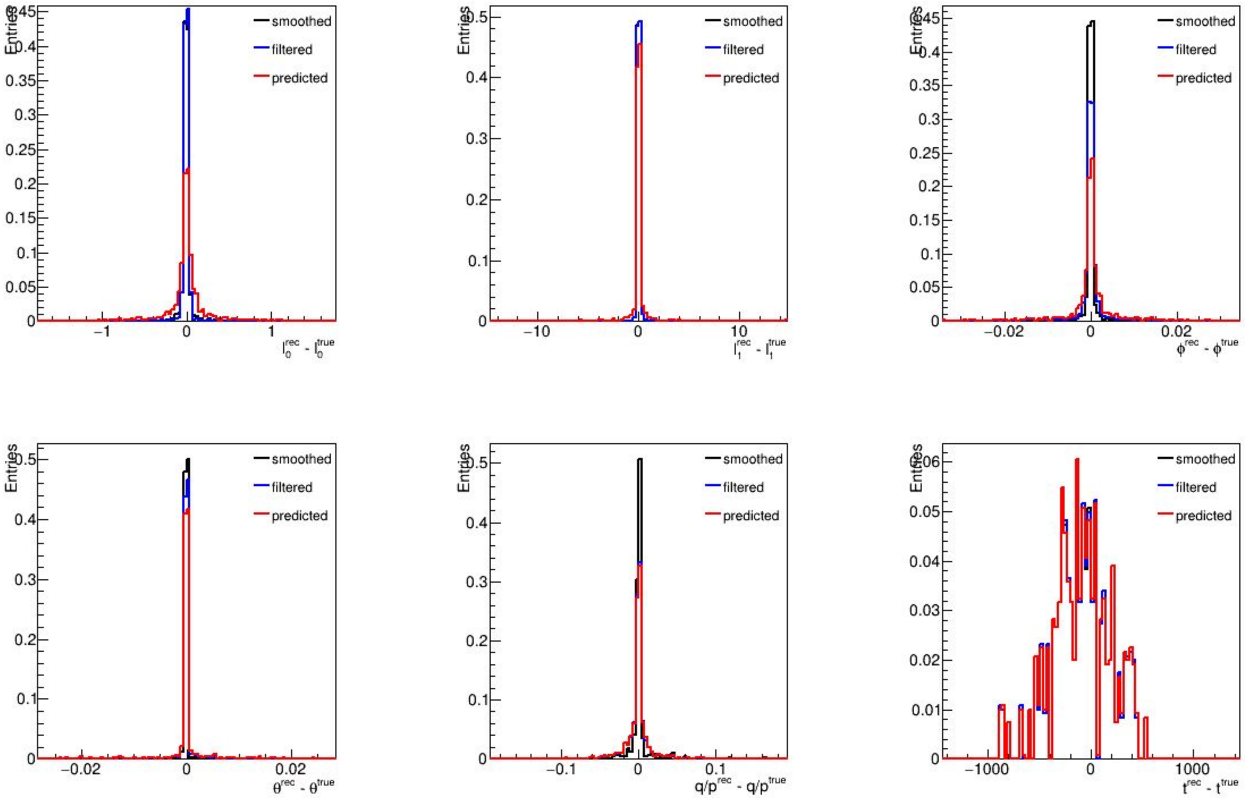


# Kalman Filter Performance (muons) - Regional pull plots



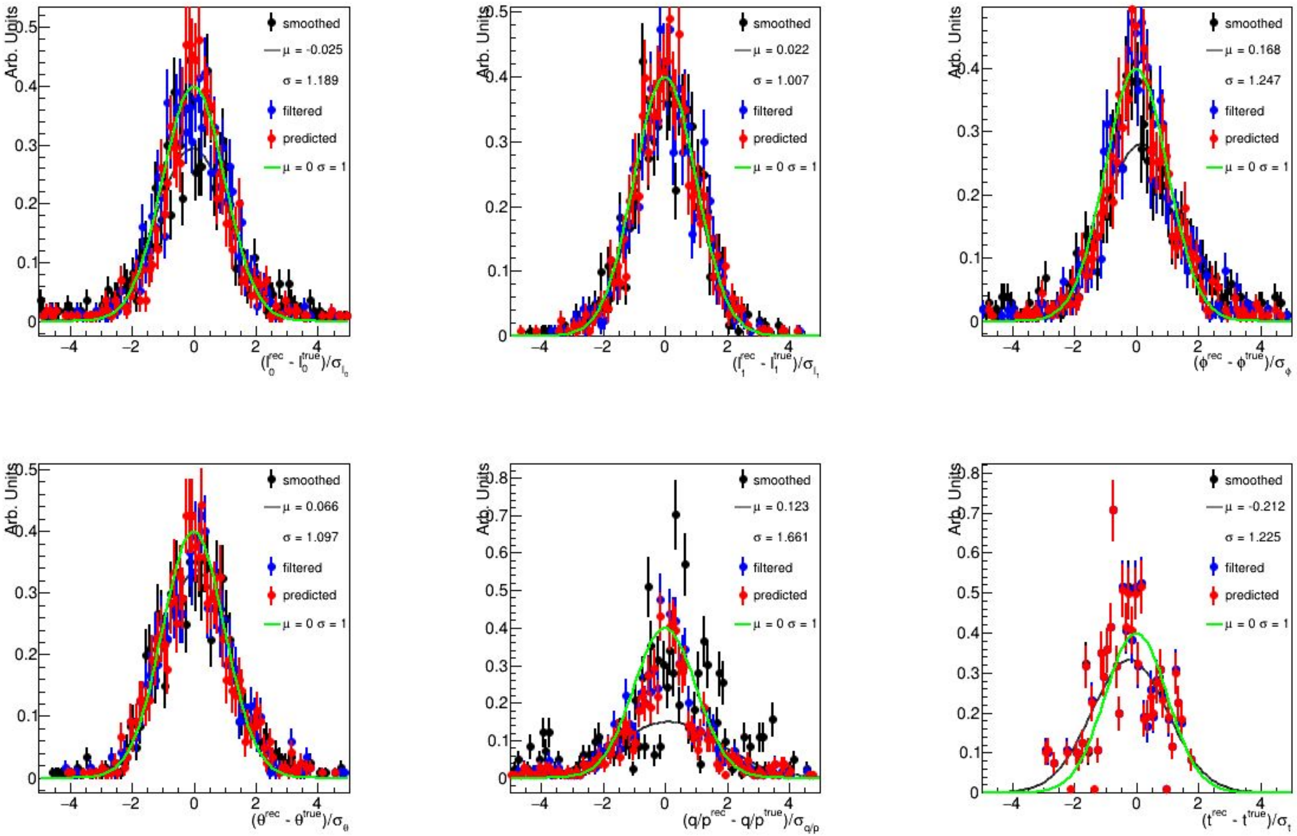
# CKF Performance (electrons) - Residual plots

Using particle gun with the same setup shown in the main presentation




note wider distributions if compared to muon reconstruction (slide 19) and the higher energy residual compared to GSF (slide 22)

# CKF Performance (electrons) - Pull Plots



# Bremsstrahlung and the Gaussian Sum Filter

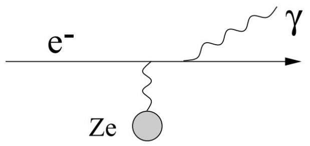
- Charged particles can lose energy by radiating electromagnetic quanta, predominantly in the Coulomb field of the nucleus.

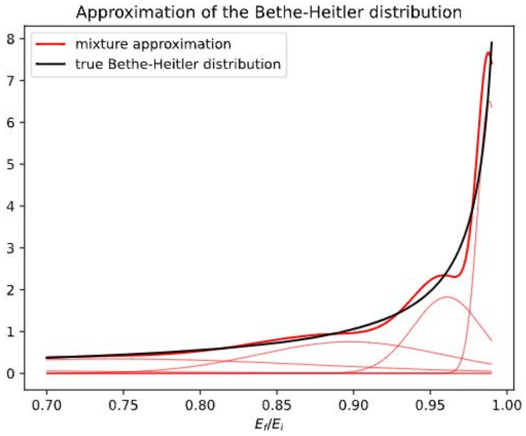
- “The characteristic E/m2 dependence is the reason that for energies below some 100 GeV energy loss through bremsstrahlung is only significant for electrons and positrons.”[5]

- To handle the non-Gaussian errors introduced by this effect, the Gaussian Sum Filter is used

- GSF is an extension of the Kalman Filter where the track state is modelled by a Gaussian mixture

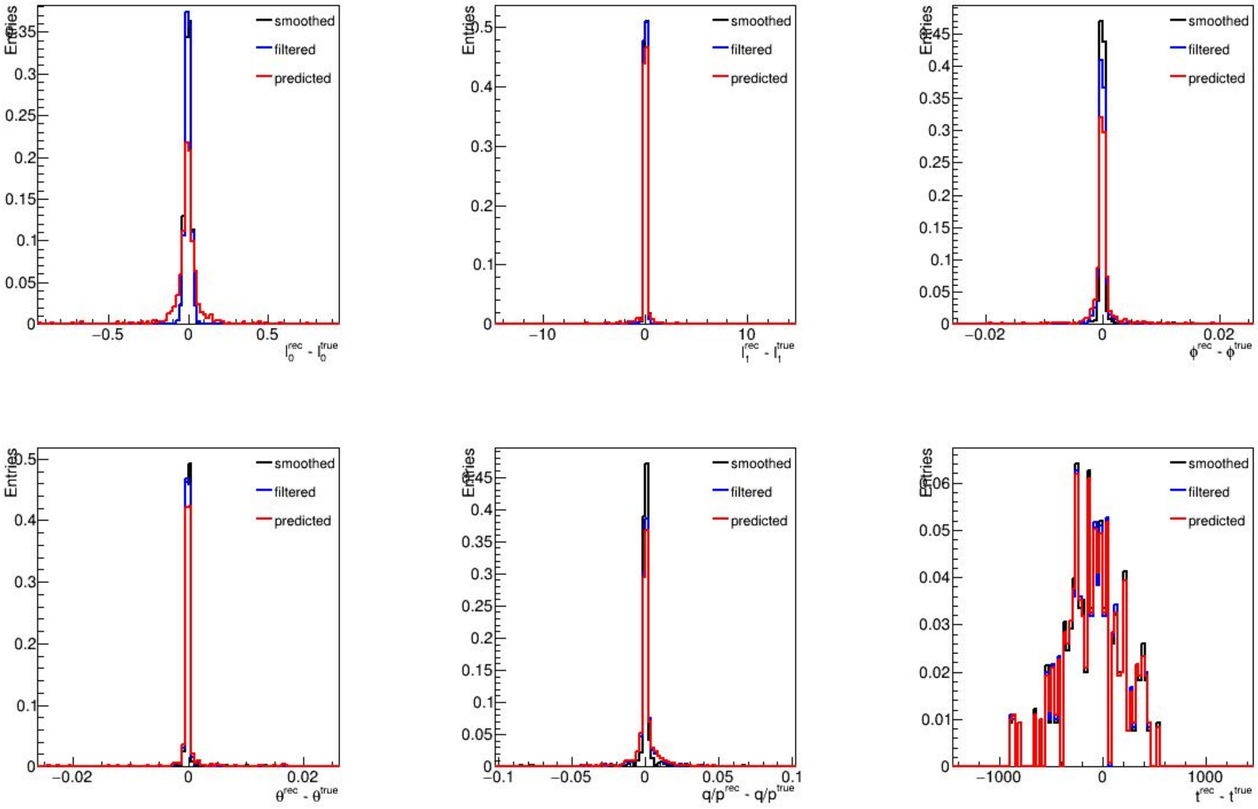
$$p(\vec{x}) = \sum_i w_i \varphi(\vec{x}; \mu_i, \Sigma_i), \quad \sum_i w_i = 1$$



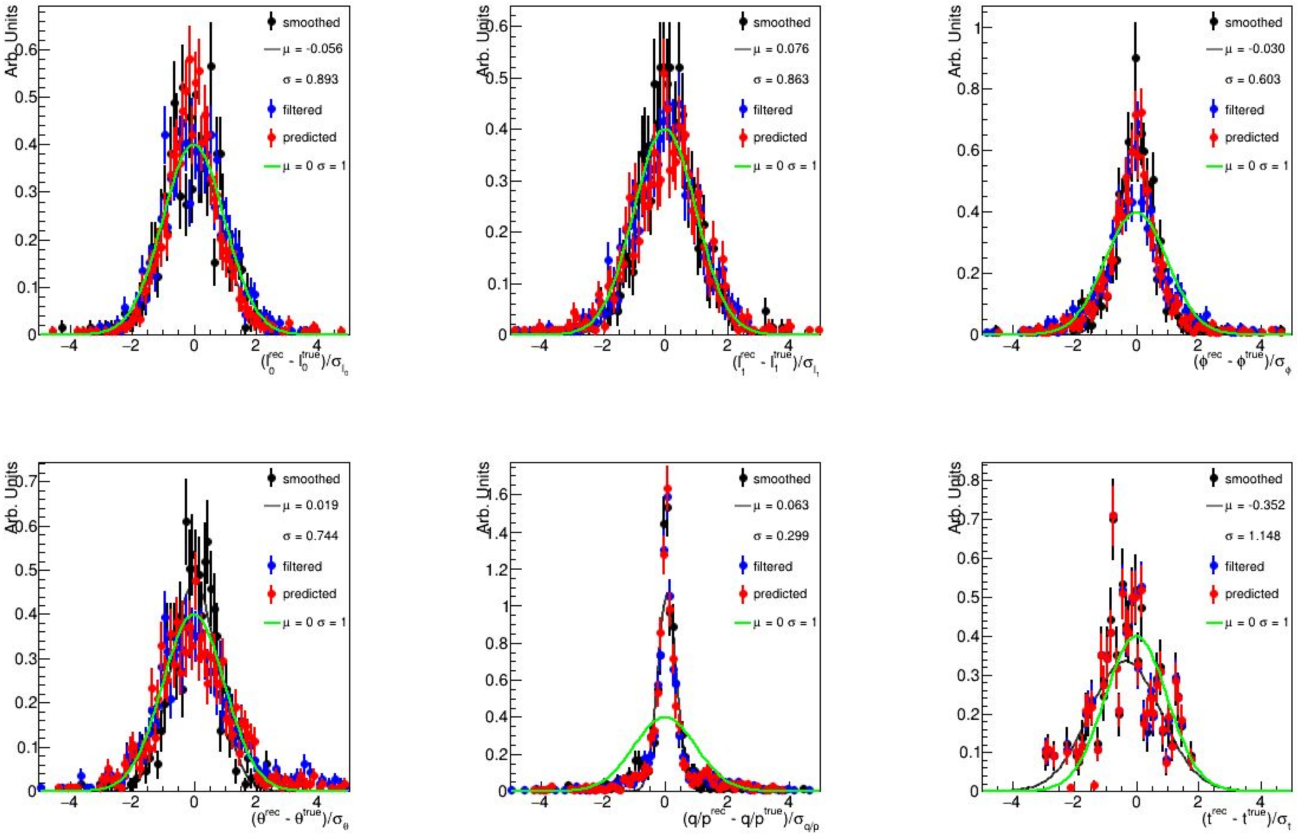




$$f(z) = \frac{(-\ln z)^{c-1}}{\Gamma(c)}, \quad c = t/\ln 2$$

# Gaussian Sum Filter performance (electrons) - Residual plots



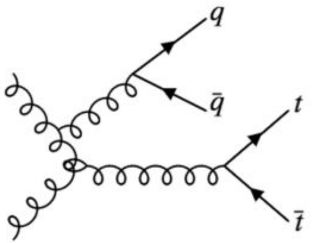
# GSF Performance (electrons) - Pull Plots



# Simulating collision events

# Simulating collision events

- Using Pythia8 we can generate Monte Carlo collision events with customizable conditions
  - Pythia documentation
  - List of hard process that can be generated
- For now simulating Top: qq’->tt’
- Are there any better events to simulate?




```
addPythia8(
    s,
    hardProcess=["Top:qqbar2ttbar=on"],
    npileup=200,
    vtxGen=acts.examples.GaussianVertexGenerator(
        mean=acts.Vector4(0, 0, 0, 0),
        stddev=acts.Vector4(0.0125 * u.mm, 0.0125 * u.mm, 55.5 * u.mm, 5.0 * u.ns),
    ),
    rnd=rnd,
    outputDirRoot=outputDir,
)
```

# Sequencer Setup - Simulation

```python
addFatras(
    s,
    trackingGeometry,
    field,
    preSelectParticles=ParticleSelectorConfig(
        rho=(0.0, 24 * u.mm),
        absZ=(0.0, 1.0 * u.m),
        eta=(-3.0, 3.0),
        pt=(150 * u.MeV, None),
        removeNeutral=True,
    ),
    rnd=rnd,
    enableInteractions=True,
)

addDigitization(
    s,
    trackingGeometry,
    field,
    digiConfigFile=digiConfigFile,
    rnd=rnd,
)
```

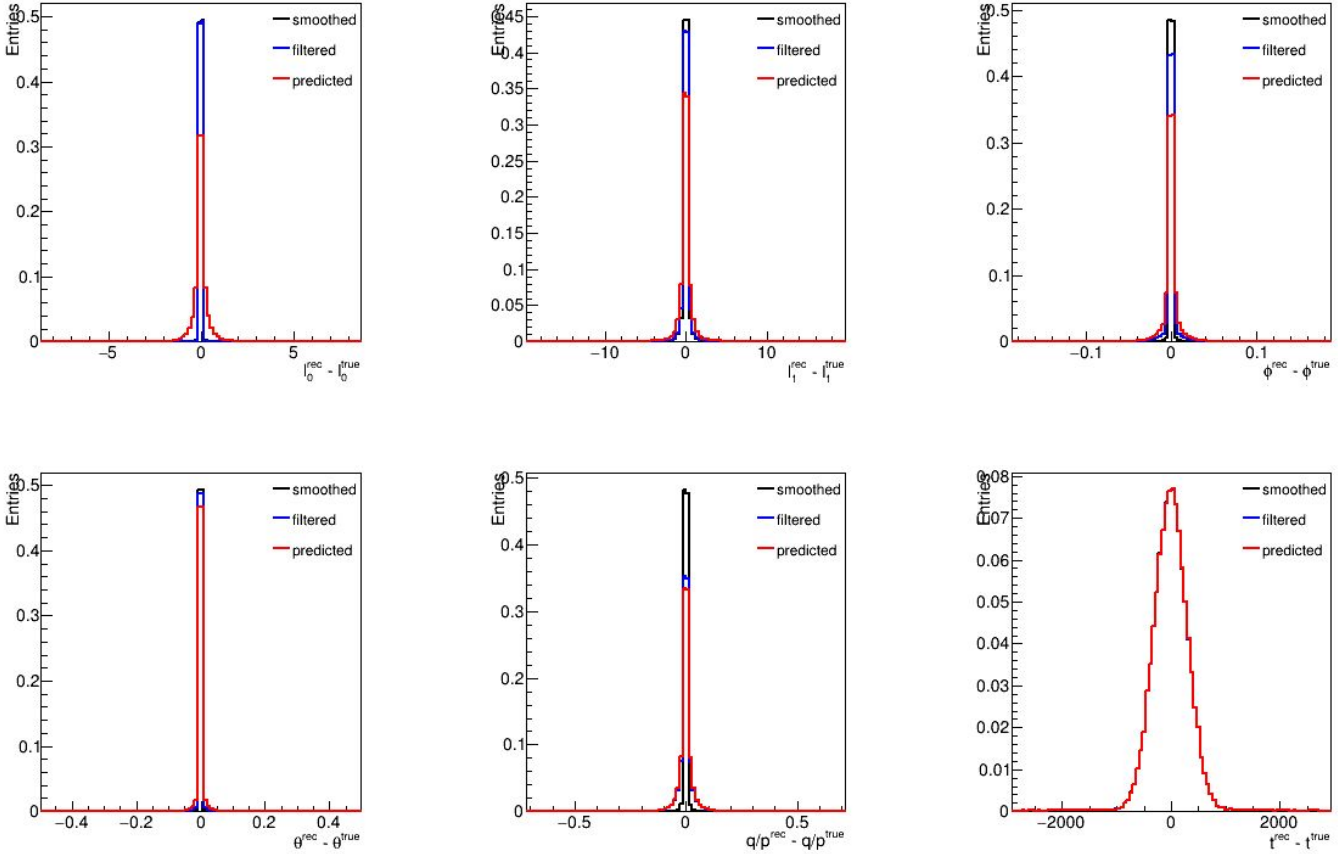
- Same as before just preselecting which particles to propagate

# Sequencer Setup - Reconstruction

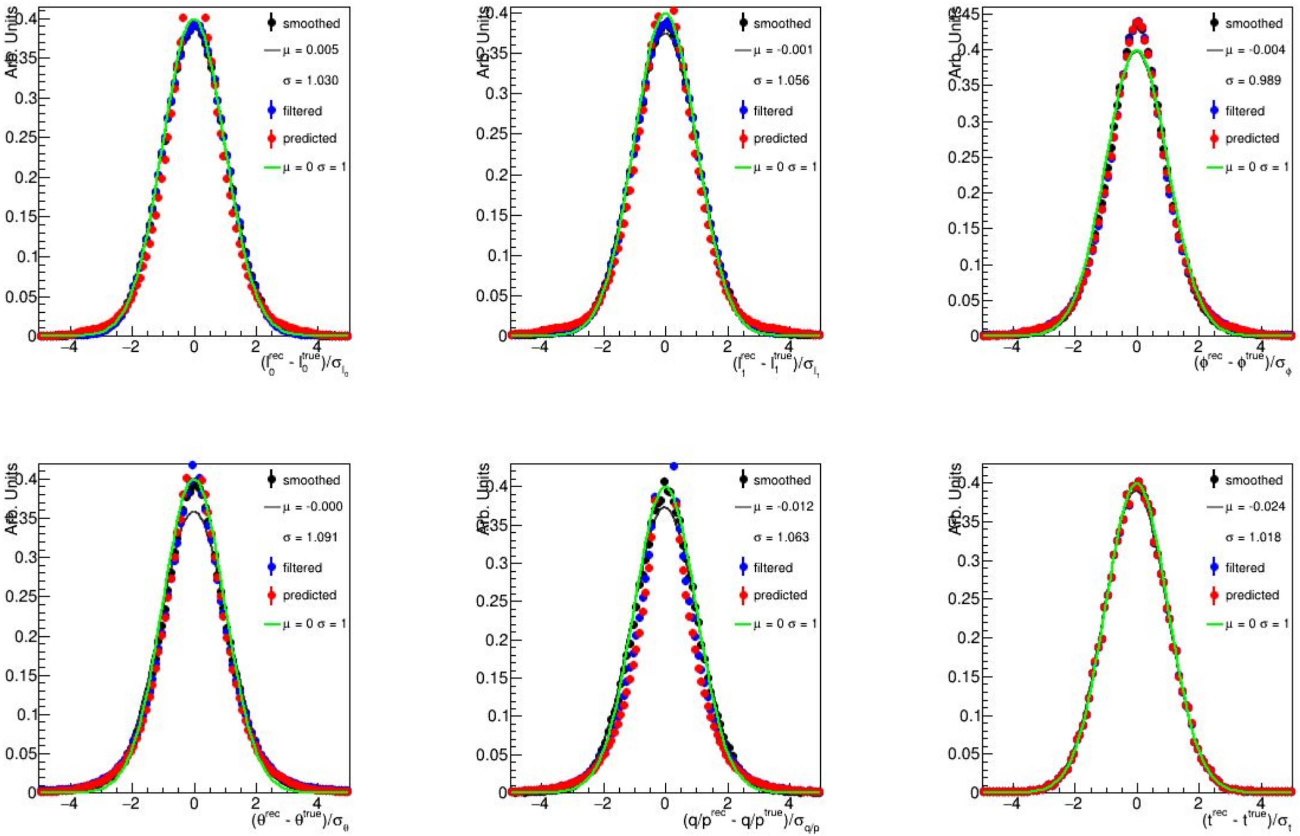
```python
addSeeding(
    s,
    trackingGeometry,
    field,
    rnd=rnd,
    inputParticles="particles_input",
    seedingAlgorithm=SeedingAlgorithm.TruthSmeared,
    truthSeedRanges=TruthSeedRanges(
        pt=(1 * u.GeV, None),
        eta=(-3.0, 3.0),
        nHits=(9, None),
    ),
    outputDirCsv= str(outputDir / "Seeding")
)

addCKFTracks(
    s,
    trackingGeometry,
    field,
    TrackSelectorConfig(
        pt=(1.0 * u.GeV,None),
        absEta=(None, 3.0),
        loc0=(-4.0 * u.mm, 4.0 * u.mm),
        nMeasurementsMin=7,
    ),
    outputDirRoot=outputDir,
    writeCovMat=True,
    outputDirCsv=outputDir,
)
```

- Now in the seeding we also select the range of truth seeding in order to make a fair performance evaluation
- *CKFTracks()* is both the fitter and the finder, here we also define some criteria to the selector

# Performance in collision event - Residual plots



# Performance in collision event - Pull plots



# Performance in collision event - Regional pull plots