ACTS First Steps

Rodrigo Estevam de Paula

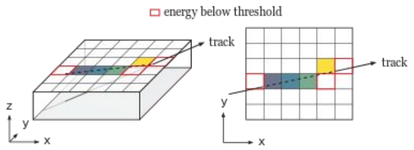
February 29, 2024

Outline

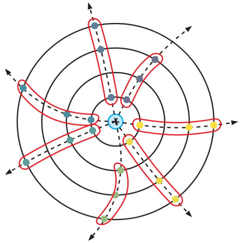
- Short tracking recap
- What is ACTS?
  - Summary
  - Setup
- Reconstruction of single particle events
  - Using Combinational Kalman Filter
  - Using Gaussian Sum Filter (GSF)

- More detailed overview (Personal notes)

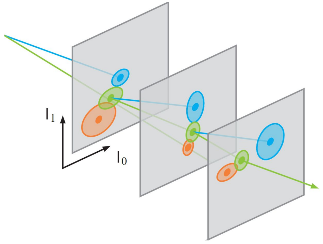
Tracking recap

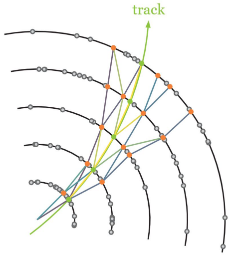


Clustering



Seeding





Track fitting and finding

Track reconstruction processing chain

Kalman Filter Summary

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

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

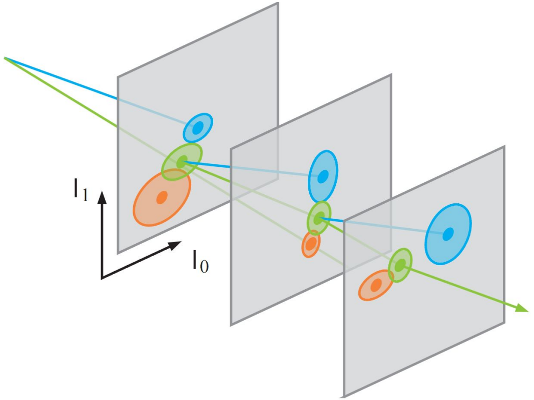


Figure 2.1: Illustration of KF estimative iteration. Measurement represented in orange, (prior) estimative in blue and filtered (posteriori) estimative in green.

- 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.

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:

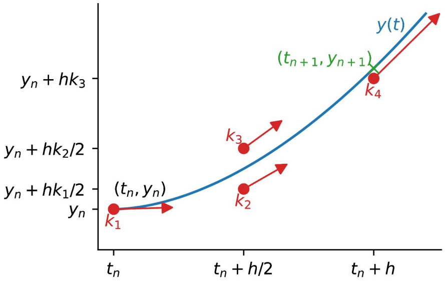
$$\frac{dy}{dt} = f(t, y), \qquad y(t_0) = y_0,$$

- 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:

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



- 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 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++”
- 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

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

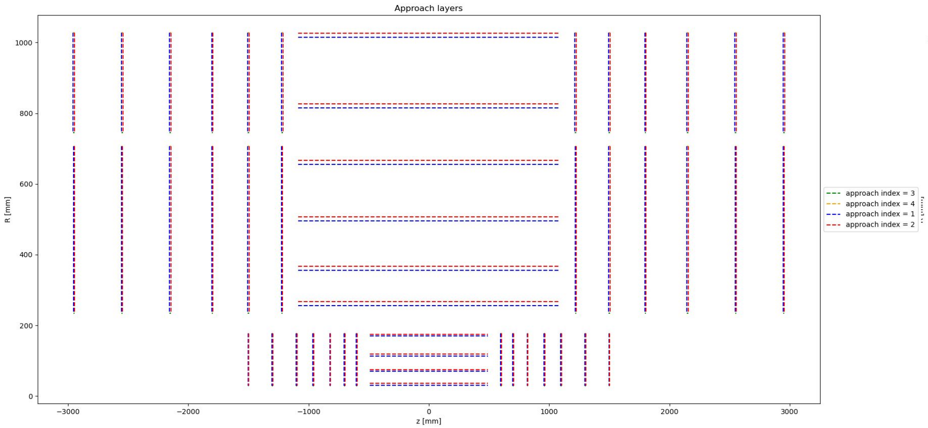
```
git clone https://github.com/acts-project/acts <source>
cd <source>
source CI/setup_cvmfs_lcg.sh
cmake -B build -S . -DACTS_BUILD_FATRAS=on -DACTS_BUILD_EXAMPLES_PYTHON_BINDINGS=ON
cmake --build build
```

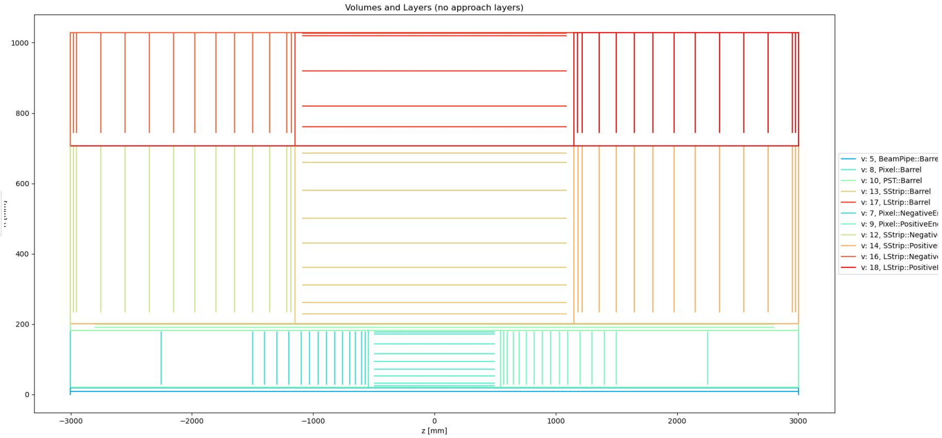
- To use Python bindings, it is also necessary to setup a Python env

```
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 simulation

Simulating events with particle guns

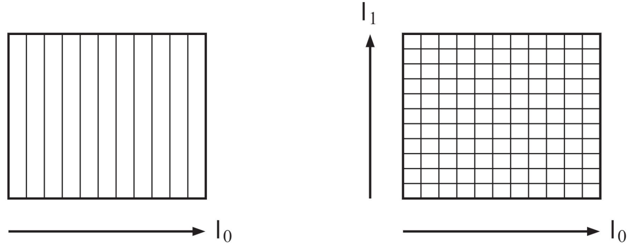
- ACTS offers a series of examples of Python bindings that can be used to simulate basic scenarios
  - The use of these bindings in “production” is encouraged
- We are going to use *Examples/Scripts/Python/truth_tracking_kalman.py* to simulate CKF
- The Setup will be
  - ODD detector structure
  - 100 muons distributed uniformly between eta -3 and 3
  - Using standard seeding algorithms
  - Using CKF to reconstruct the tracks

```python
if "__main__" == __name__:
    srcdir = Path(__file__).resolve().parent.parent.parent.parent

    detector, trackingGeometry, decorators = acts.examples.GenericDetector.create()

    field = acts.ConstantBField(acts.Vector3(0, 0, 2 * u.T))

    runTruthTrackingKalman(
        trackingGeometry,
        field,
        digiConfigFile=srcdir
        / "Examples/Algorithms/Digitization/share/default-smearing-config-generic.json",
        outputDir=Path.cwd(),
    ).run()
```

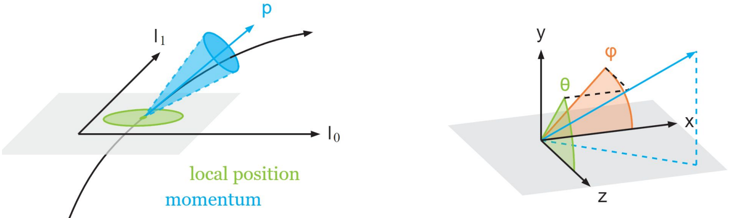
State vector and analysis

- 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)

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

```
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

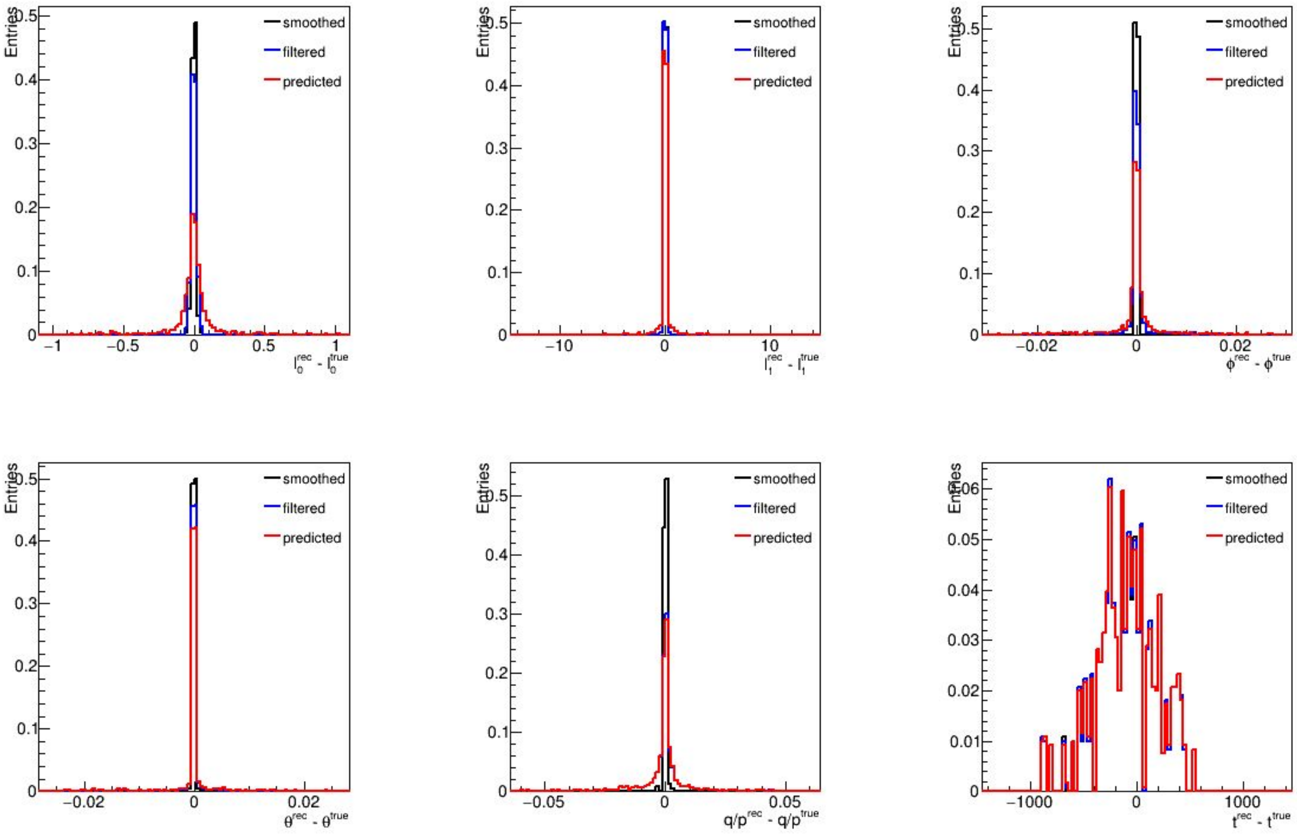
Could not turn on debug options to see the generated data :(

Fitting and Finding

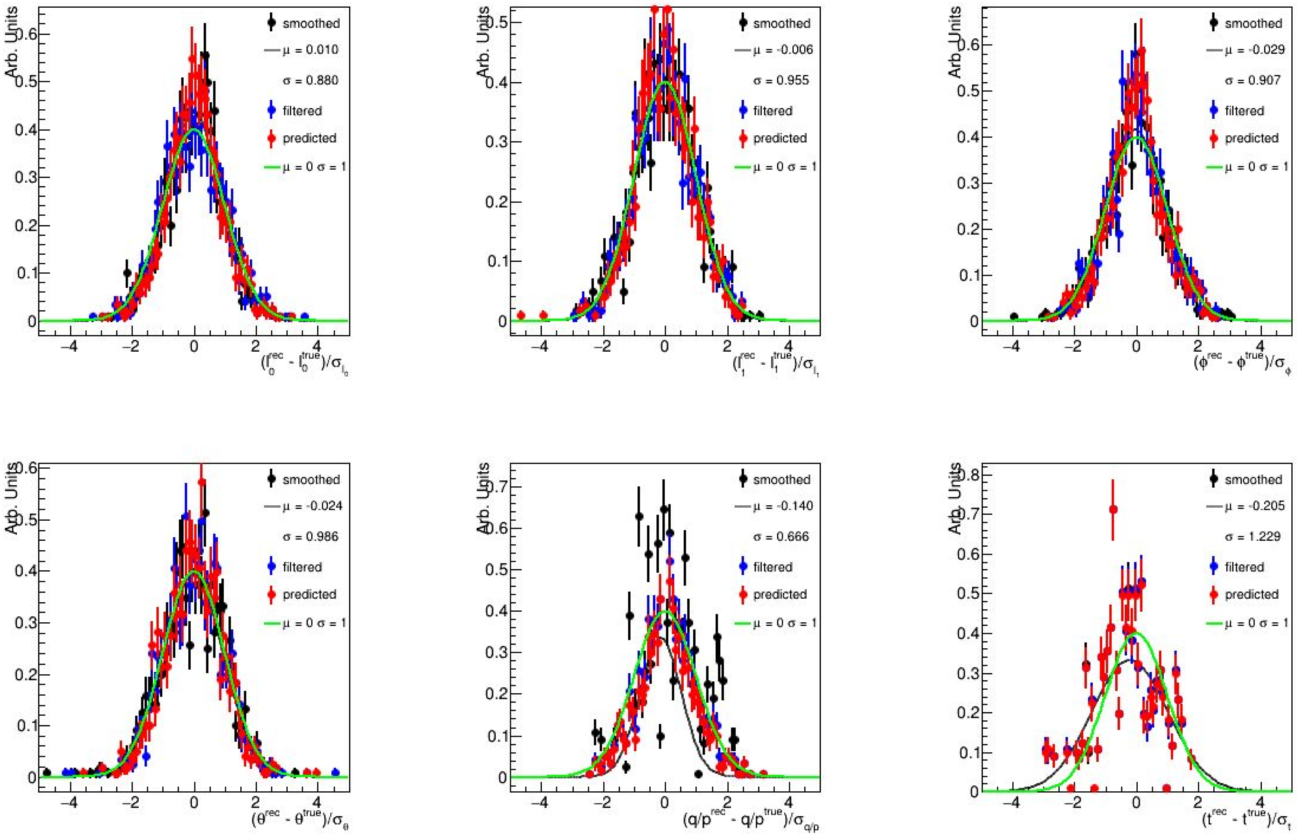
```
addKalmanTracks(
    s,
    trackingGeometry,
    field,
    directNavigation,
    reverseFilteringMomThreshold,
)

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

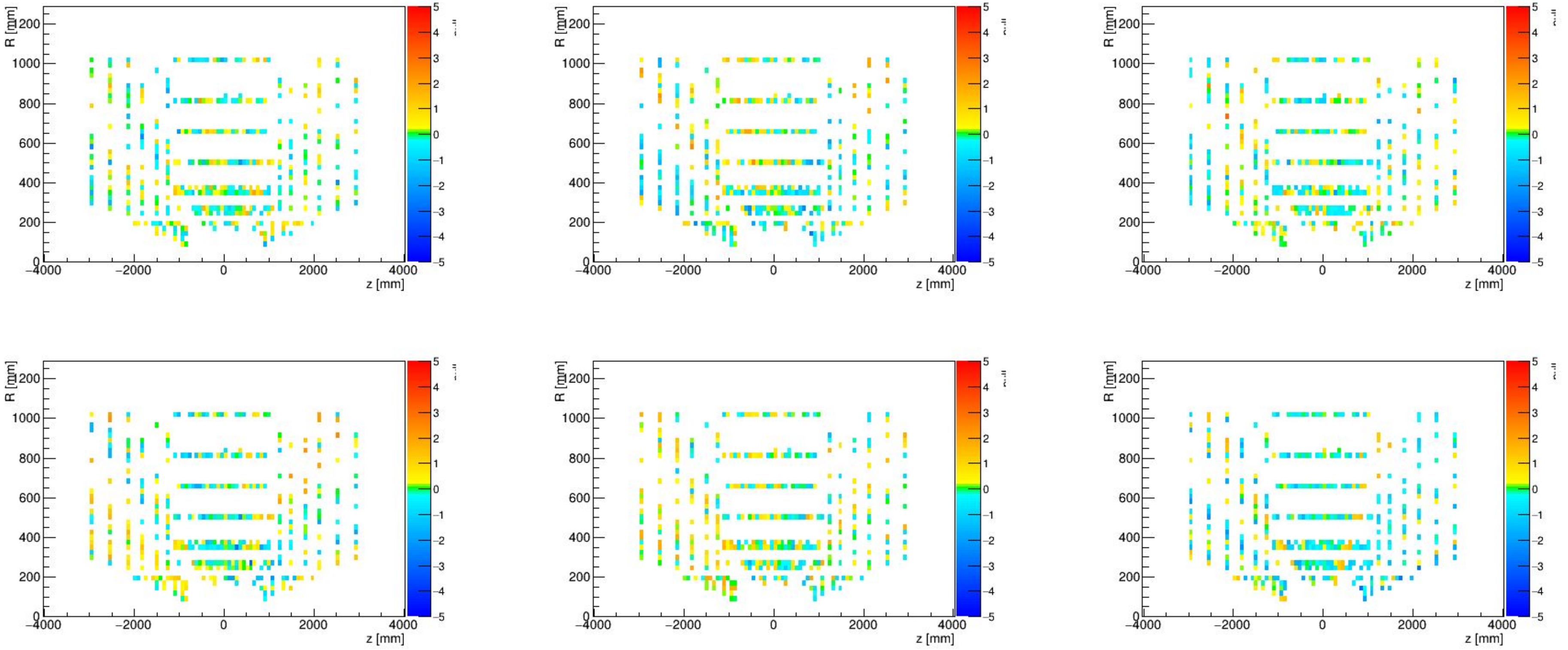
Kalman Filter performance (muons) - Residual plots



Kalman Filter performance (muons) - Pull plots



Kalman Filter Performance (muons) - Regional pull plots



Bremsstrahlung and the Gaussian Sum Filter

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

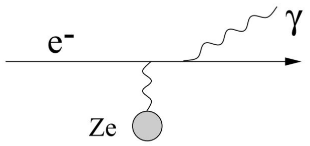
- "The characteristic E/m2 dependence is the reason that for energies below some 100 GeV energy loss through bremsstrahlung is only significant for electrons and positrons."*

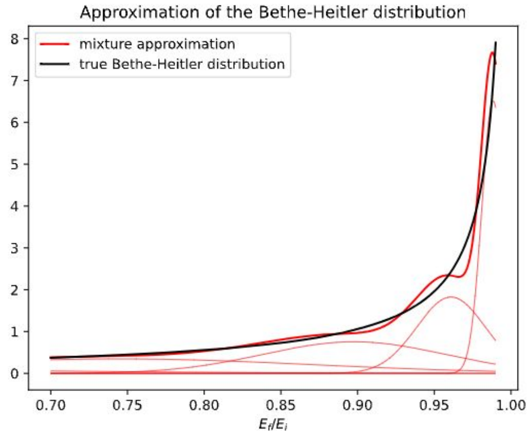
- To handle the non-Gaussian errors introduced by this effect, the Gaussian Sum Filter is used

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

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

- Is also necessary to adapt the Extrapolation method (slide 5)

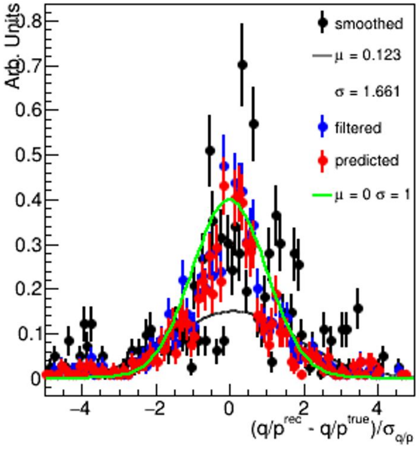




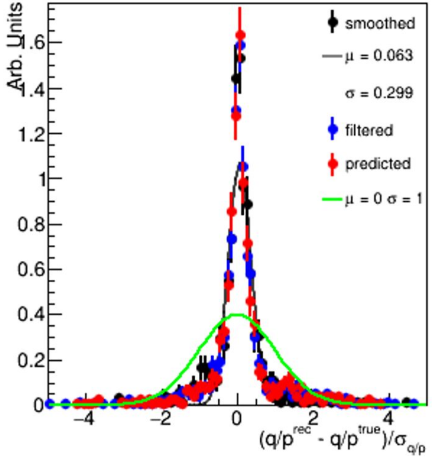
$$f(z) = \frac{(-\ln z)^{c-1}}{\Gamma(c)}, \quad c = t/\ln 2$$

CKF and GSF comparison

- Test done with
  - Particle gun of electrons uniformly between eta -3 and 3
  - Same seeding configuration
- The pull performance for positional parameters is similar and is not obvious that the GSF is better
- The energy is the parameter where the difference is greater (plots in the right)
- GSF is way slower than the CKF (using same setup)



CKF



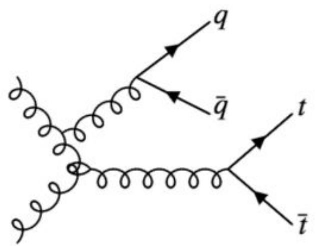
GSF

-

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

```
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,
)
```

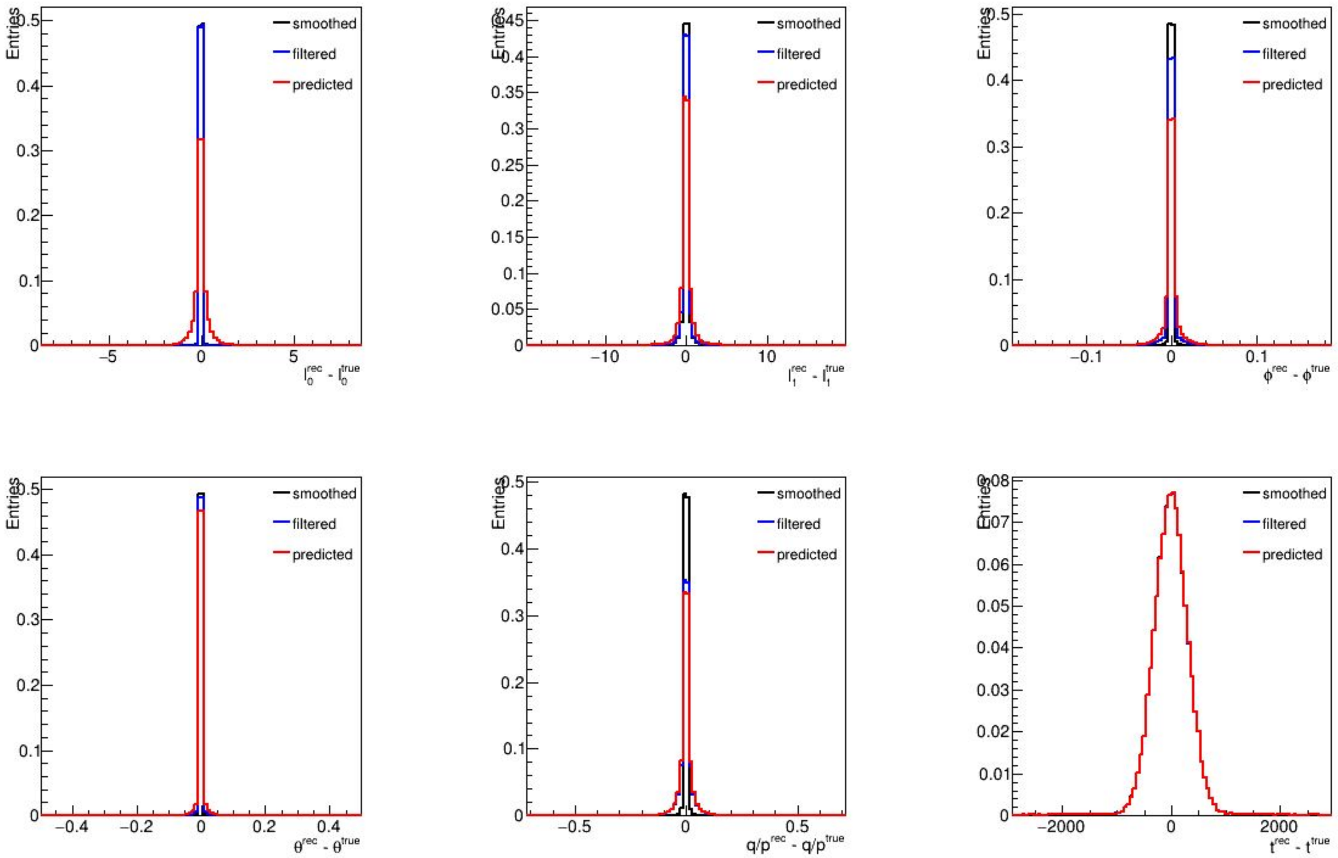
- Same as before just preselecting which particles to propagate

Sequencer Setup - Reconstruction

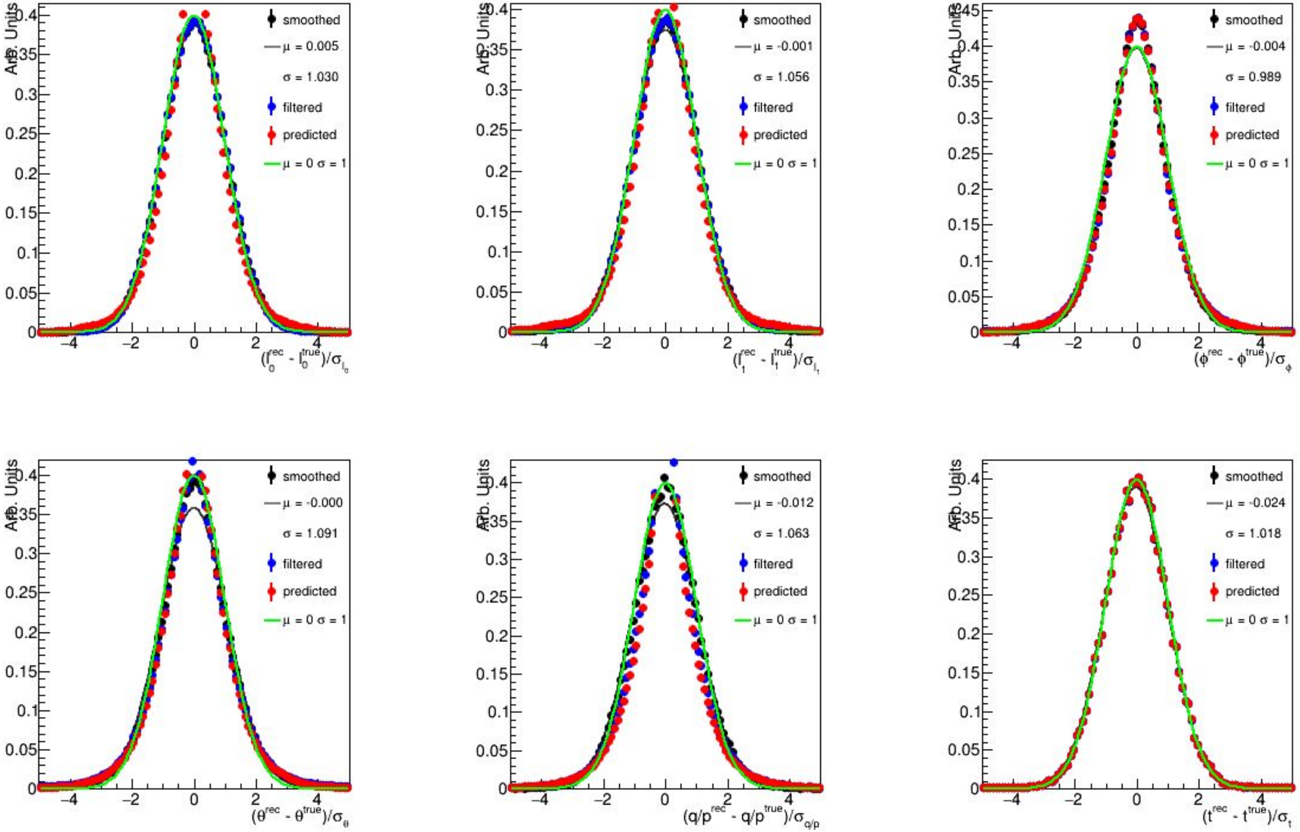
```
addSeeding(
    s,
    trackingGeometry,
    field,
    rnd=rnd,
    inputParticles="particles_input",
    seedingAlgorithm=SeedingAlgorithm.TruthSmeared,
    truthSeedRanges=TruthSeedRanges(
        pt=(1 * u.GeV, None),
        eta=(-3.0, 3.0),
        nHits=(9, None),
    ),
    outputDirCsv=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

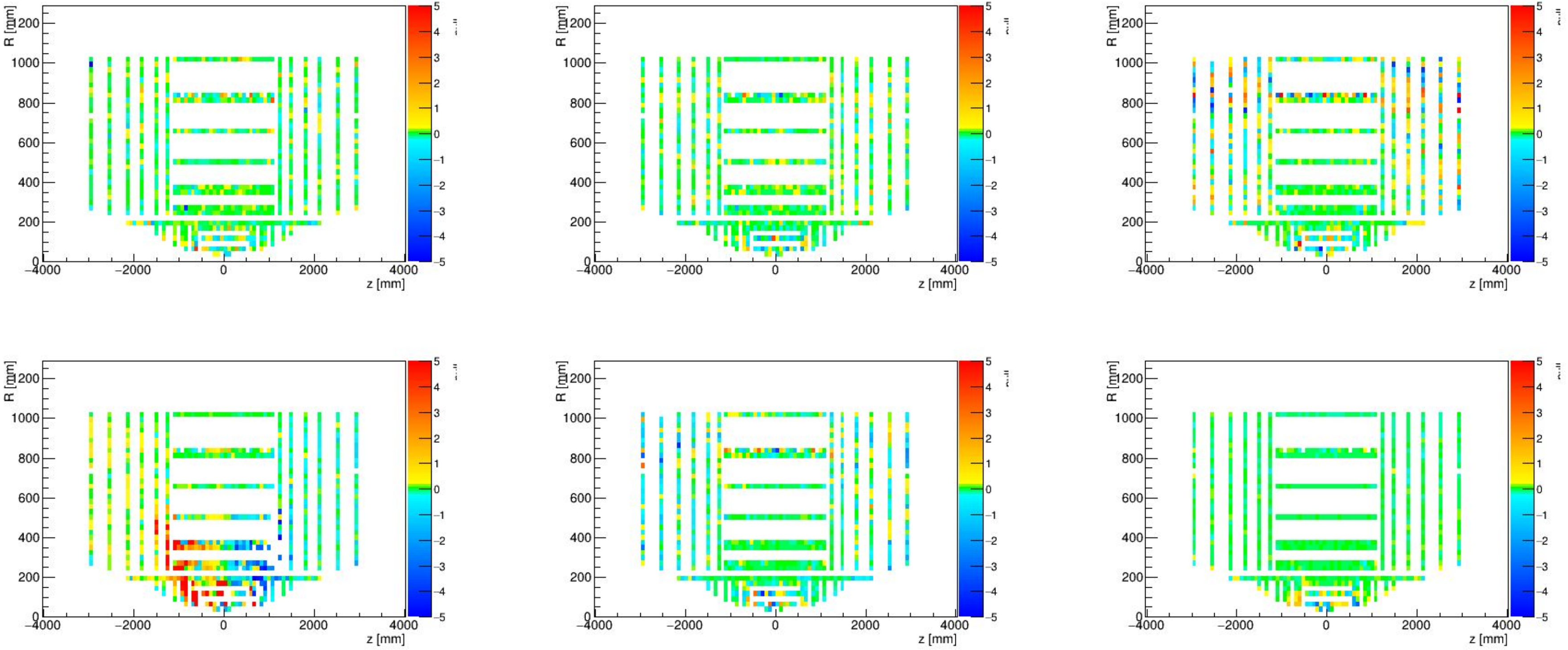
Performance in collision event - Residual plots



Performance in collision event - Pull plots



Performance in collision event - Regional pull plots



Next steps

Explore the ACTS track reconstruction framework (on going)

- **Next ?:** Study the implementation of the CKF and GSF in the core library
  - Hope to present next meeting
- How to add a custom algorithm in ACTS?
- Get the ITk and HGTD geometry to work in the ACTS

Follow HGTD ACTS integration campaign

Theoretical Study

- H. Kolanosky, Particle Detectors (2020)
  - Finished Chapters 1-4
  - Next: Chapters 5 and 8-9
- Some book about GNNs

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

Backup