Implementation of Large Radius tracking in Acts

By

MATTOUHI AICHA

A report submitted as part of the summer student program
in ATLAS experiment at Cern

23 August 2024

Supervisor(s) Doga Elitez and Paul Gessinger

ABSTRACT

In high-energy physics, efficient reconstruction of particle trajectories is crucial for accurate analysis and understanding of complex events. This project focuses on optimizing large-radius tracking using the fictitious Open Data Detector (ODD) ,a detector system used to test particle tracking algorithms. We developed a new vertex generator tailored specifically for these types of particles. The new vertex generator is integrated into the existing framework to improve path reconstruction for large production radius cases. We systematically evaluate its performance by comparing tracking efficiency metrics with the ODD example.

ACKNOWLEDGMENTS

I would like to express my deepest gratitude to my supervisors, Doga Elitez and Dr. Paul Gessinger, for their invaluable support and guidance throughout this project. Their expertise and mentorship have not only shaped the direction of this work but have also enriched my understanding of the field. Thank you for defining this project and for teaching me so much along the way.

I am also profoundly grateful to CERN for providing me with this once-in-a-lifetime opportunity. The experience has been both enriching and inspiring, allowing me to grow both professionally and personally.

Lastly, I want to extend my heartfelt thanks to my family and friends. Your belief in me and your continuous support have been my foundation. Thank you for always being there, encouraging me, and helping me every step of the way

Contents

List of Figures

Chapter 1

Introduction

Reconstructing the paths of charged particles is a crucial but very demanding task in high-energy physics. As future colliders like the High-Luminosity Large Hadron Collider (HL-LHC) and the Future Circular Collider (FCC) are designed to be more advanced, they will create more particle collisions at once, making tracking even harder due to this "pile-up" effect. To keep up with these challenges, it's needed to develop new algorithms that can take full advantage of modern computing power, including multi-core processors and specialized hardware.

ACTS (A Common Tracking Software) project aims to tackle this issue. Building on the experience gained from the ATLAS experiment at the LHC, ACTS aims to create a versatile software library that works with any experiment setup and is optimized for current computing technology. This library will include advanced tracking techniques developed at the LHC while being adaptable to different experimental conditions.

Our focus is on optimising the Acts library for Large Radius Tracking (LRT) using the Open Data Detector (ODD). The ODD is a sophisticated test detector that simulates the environment of the HL-LHC and mimics the design of the ATLAS Inner Tracker (ITk) upgrade. By testing track reconstruction algorithms with this realistic setup, we can ensure that our improvements will be effective for actual detectors (Paul Gessinger-Befurt, 2023) .

Chapter 2

Large Radius Tracking

2.1 Definition

Large Radius Tracking (LRT) refers to the process of tracking particles that travel long distances within a detector before interacting with its components. In high-energy physics experiments, most particles produced in collisions have relatively short paths before they decay or interact. However, some particles, particularly those with longer lifetimes or lower momenta, can travel significant distances from the collision point before interacting with the detector material.

LRT focuses on accurately identifying and reconstructing the trajectories of these long-lived particles, which typically have larger impact parameters (the perpendicular distance between the particle's path and the collision point) compared to more common short-lived particles (see Figure 2.1). This type of tracking is crucial for studying phenomena such as rare decays, beyond Standard Model particles, or particles with delayed decays that may evade detection by standard tracking algorithms optimized for shorter paths.

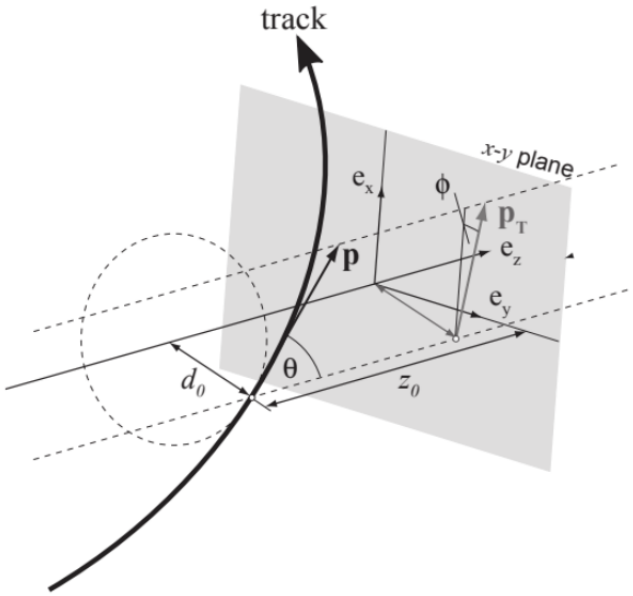

Figure 2.1: Coordinates in tracking

2.1.1 Tracking in ACTS

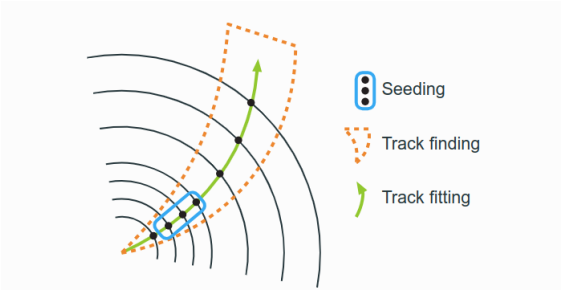

Figure 2.2: Tracking in Acts

Chapter 3

New Vertex Generator

3.1 Gaussian Displaced Vertex Generator

To simulate long-lived particles (originating far from the main interaction point), a new vertex generator was created.

Listing 3.1: Gaussian Displaced Vertex Position Generator

```
struct GaussianDisplacedVertexPositionGenerator
    : public EventGenerator::PrimaryVertexPositionGenerator {
  double rMean = 0;
  double rStdDev = 1;
  double zMean = 0;
  double zStdDev = 1;
  double tMean = 0;
  double tStdDev = 1;

  Acts::Vector4 operator()(RandomEngine& rng) const override {
    double min_value = -M_PI;  // -
    double max_value = M_PI;   //

    std::uniform_real_distribution<> uniform(min_value, max_value);

    std::normal_distribution<double> rDist(rMean, rStdDev);
    std::normal_distribution<double> zDist(zMean, zStdDev);
```

```
18      std::normal_distribution<double> tDist(tMean, tStdDev);
19
20      // Generate random values from normal distributions
21      double r = rDist(rng);
22      double phi = uniform(rng);  // Random angle in radians
23      double z = zDist(rng);
24      double t = tDist(rng);
25
26      // Convert cylindrical coordinates to Cartesian coordinates
27      double x = r * std::cos(phi);
28      double y = r * std::sin(phi);
29
30      return Acts::Vector4(x, y, z, t);
31    }
32 };
```

This was done using cylindrical coordinates by generating the radius r from a centered Gaussian distribution and the angle ϕ from a uniform distribution ranging from $-\pi$ to π, and then converting to Cartesian coordinates.

3.2 Python Bindings

This code binds the GaussianDisplacedVertexPositionGenerator C++ class to Python, allowing it to be instantiated and its attributes (rMean, rStdDev, zMean, zStdDev, tMean, and tStdDev) to be directly accessed and modified in Python. It provides both a default constructor and a custom constructor that initializes these attributes with specified values. This is done because the examples we have in the ACTS. library are written in python .

Listing 3.2: python bindings for the new vertex generator

```
py::class_<
      ActsExamples::GaussianDisplacedVertexPositionGenerator,
      ActsExamples::EventGenerator::PrimaryVertexPositionGenerator,
      std::shared_ptr<ActsExamples::
         GaussianDisplacedVertexPositionGenerator>>(
      mex, "GaussianDisplacedVertexPositionGenerator")
      .def(py::init<>())
      .def(py::init([](double rMean, double rStdDev, double zMean,
                       double zStdDev, double tMean, double tStdDev) {
            ActsExamples::GaussianDisplacedVertexPositionGenerator g;
            g.rMean = rMean;
            g.rStdDev = rStdDev;
            g.zMean = zMean;
            g.zStdDev = zStdDev;
            g.tMean = tMean;
            g.tStdDev = tStdDev;
            return g;
          }),
          py::arg("rMean"), py::arg("rStdDev"), py::arg("zMean"),
          py::arg("zStdDev"), py::arg("tMean"), py::arg("tStdDev"))
      .def_readwrite(
          "rMean",
          &ActsExamples::GaussianDisplacedVertexPositionGenerator::rMean)
      .def_readwrite(
          "rStdDev",
          &ActsExamples::GaussianDisplacedVertexPositionGenerator::rStdDev
             )
      .def_readwrite(
          "zMean",
          &ActsExamples::GaussianDisplacedVertexPositionGenerator::zMean)
      .def_readwrite(
```

```
30            "zStdDev",
31            &ActsExamples::GaussianDisplacedVertexPositionGenerator::zStdDev
                  )
32        .def_readwrite(
33            "tMean",
34            &ActsExamples::GaussianDisplacedVertexPositionGenerator::tMean)
35        .def_readwrite(
36            "tStdDev",
37            &ActsExamples::GaussianDisplacedVertexPositionGenerator::tStdDev
                  );
```

3.3 Results of Displaced Vertex Generator

Here in Figure 3.1, we can see a plot resulting from running the ttbar example using the new displaced vertex position generator. The color scale on the right side of the graph ranges from dark blue to yellow, representing the density of entries in each bin of the histogram. There is clear symmetry along both axes in the number of particles at each vertex.We can see that vertices are forming a donut shape as expected in order to have a region of interest for particles with a certain production radius

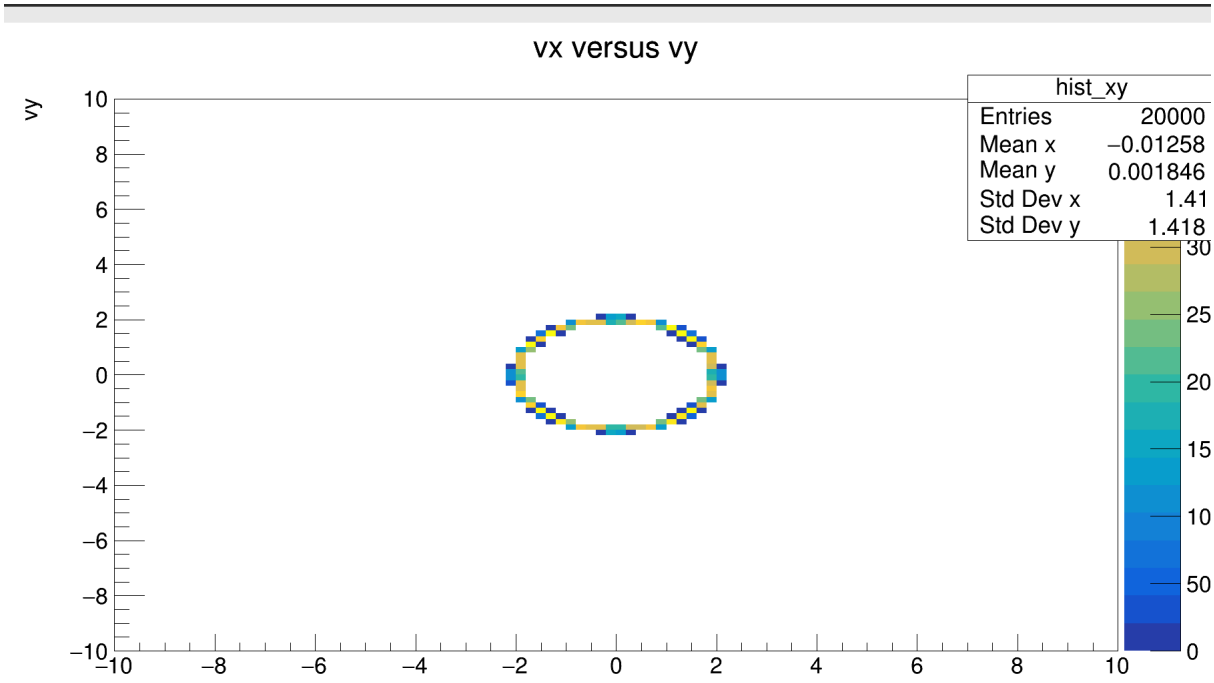

Figure 3.1: Histogram of generated vertices using the new vertex generator

Chapter 4

Efficiency

4.1 Efficiency in ROOT

The data analysis in this project, as in most projects at CERN, is conducted using the ROOT framework. ROOT is a powerful software toolset developed at CERN, specifically designed for data analysis in high-energy physics. It provides comprehensive tools for the storage, processing, and visualization of large datasets. With features for statistical analysis, data fitting, histograms, and graphing, ROOT is highly optimized to manage the enormous volumes of data generated in particle physics experiments. It is widely utilized for analyzing collision data from the Large Hadron Collider (LHC) and other similar experiments.

This Efficiency object in Root handles the calculation of efficiencies and their uncertainties. It provides several statistical methods for calculating frequentist and Bayesian confidence intervals, as well as a function for combining multiple efficiencies .

4.1.1 Comparison using Full Chain ODD

The Full Chain ODD is a top-level example script that tests the tracking of particles generated in the Open Data Detector. While , the LRT example, created during this project, focuses on tracking long-lived particles in the Open Data Detector, with the goal of establish-

ing a dedicated pass for these types of particles Here, in this plot , we can see that efficiencies for both LRT and ODD are similar, and this is because no special configuration was made for LRT.

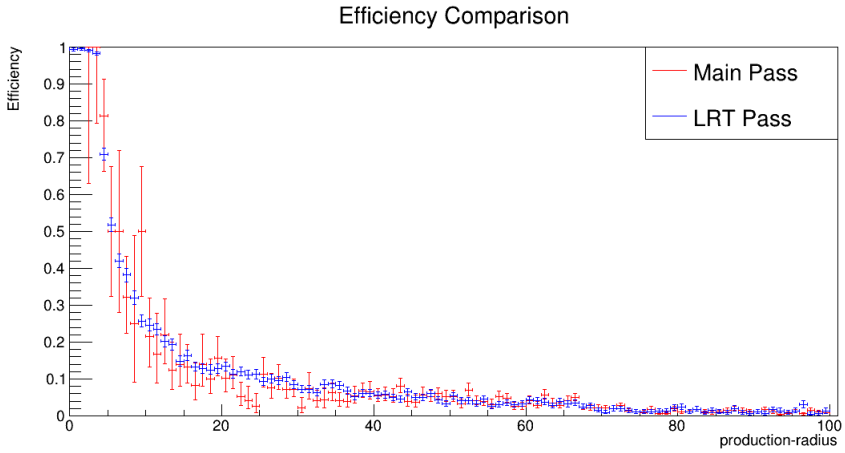

Figure 4.1: Comparison of Efficiencies

4.2 Impact Parameters Effect on Efficiency

To implement the LRT's into ACTS, not only is a new vertex generator required, but modifications to the seeding and track-finding configurations are also necessary to improve efficiency at large production radii. Here, we focus on adjusting the seeding algorithm configurations to optimize them for long-lived particles. Among the parameters considered, the most promising candidate for optimization is the maximum impact parameter d0. The following plots illustrate how changes in impactMax affect seeding efficiency in the context of LRT.

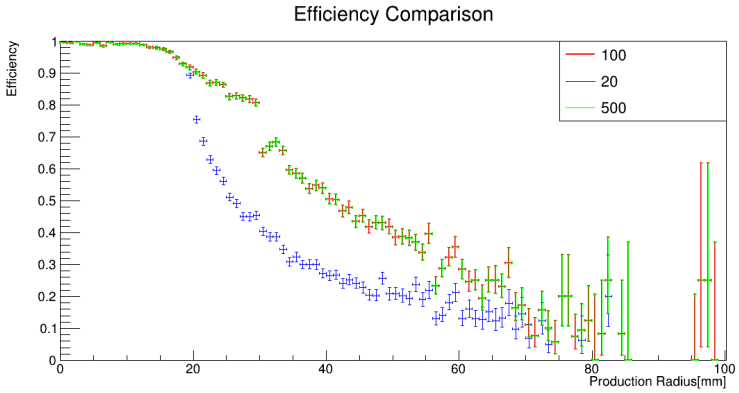

Figure 4.2: Efficiency as a function of production radius

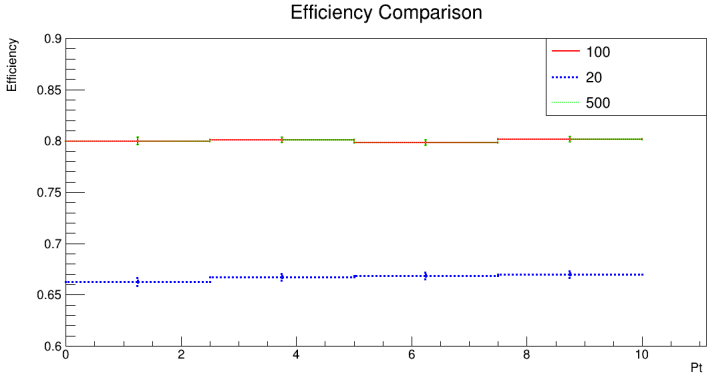

Figure 4.3: Efficiency as a function of transverse momentum (p_T)

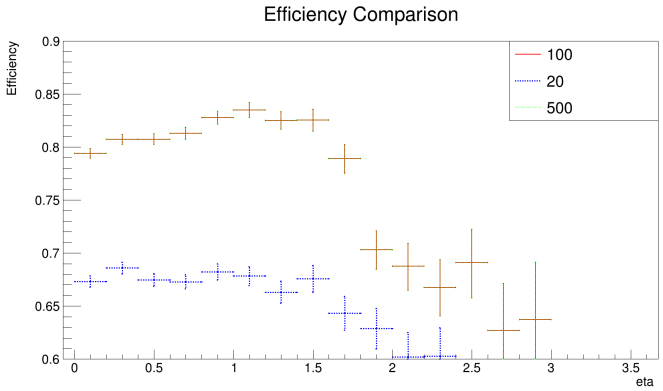

Figure 4.4: Efficiency as a function of pseudorapidity (η)

Chapter 5

CPU Comparison

5.1 Comparison of CPU Usage between Full Chain ODD and the LRT Example

Motivation: We expect the LRT pass to pick up different seeds and tracks from ODD according to its configuration, and it is important to understand how this affects computing performance.

The graph demonstrates that the CPU usage is highly variable for the LRT method compared to the ODD method. While LRT shows significant fluctuations in runtime, particularly with a sharp peak at index 2, the ODD method remains relatively stable across all the indices(measuremnts of run time) . This indicates that LRT's performance is less consistent, with occasional spikes in CPU usage, whereas ODD maintains a more predictable and steady CPU performance.

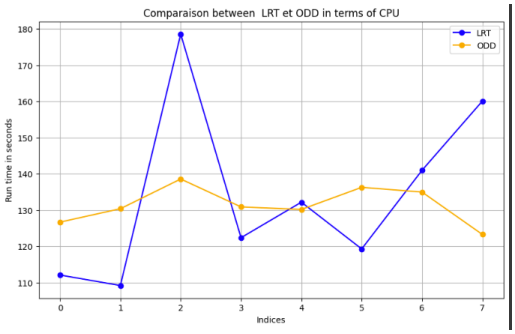

Figure 5.1: CPU comparison

Chapter 6

Conclusions

6.1 conclusion

The implementation of Large-Radius Tracking (LRT) in the ACTS framework marks an important milestone, as it is the first time LRT has been integrated into ACTS. Having two passes in ACTS (LRT and the main pass) can help improve tracking and analysis.

While there's still a need for further refinement to fully optimize the LRT configuration, the progress made so far is a good start. In this project, I focused on a specific configuration of the seeding and impact parameters, which had a positive effect on efficiency. This suggests that, with careful adjustments to other parameters, we can achieve greater efficiency in LRT.(altschul1997gapped)

References

Paul Gessinger-Befurt Andreas Salzburger, Joana Niermann (2023). "The Open Data Detector Tracking System". In: *Journal of Physics: Conference Series*.

Websites consulted

- ROOT documentation – https://root.cern.ch/root/htmldoc/guides/users-guide/Histograms.html
- ACTS repository – https://github.com/acts-project/acts/
- PYTHON BINDING Documentation – https://acts.readthedocs.io/en/latest/examples/python_bindings.html#python-based-example-scripts
- Ci-dependencies repository – https://github.com/acts-project/ci-dependencies
- ATLAS tracking software tutorial – https://atlassoftwaredocs.web.cern.ch/trackingTutorial/index.html
- Cmake Documentation – https://cmake.org/documentation/