Computing and Software for Big Science (2022) 6:8
https://doi.org/10.1007/s41781-021-00078-8

ORIGINAL ARTICLE

A Common Tracking Software Project

Xiaocong Ai[3] · Corentin Allaire[5] · Noemi Calace[5] · Angéla Czirkos[1] · Markus Elsing[5] · Irina Ene[8] · Ralf Farkas[11] · Louis-Guillaume Gagnon[8] · Rocky Garg[4] · Paul Gessinger[5] · Hadrien Grasland[6] · Heather M. Gray[8,14] · Christian Gumpert[1] · Julia Hrdinka[1] · Benjamin Huth[15] · Moritz Kiehn[5] · Fabian Klimpel[5,16] · Bernadette Kolbinger[5] · Attila Krasznahorkay[5] · Robert Langenberg[10] · Charles Leggett[14] · Georgiana Mania[3,12] · Edward Moyse[10] · Joana Niermann[5,13] · Joseph D. Osborn[9] · David Rousseau[6] · Andreas Salzburger[5] · Bastian Schlag[2,5] · Lauren Tompkins[4] · Tomohiro Yamazaki[8] · Beomki Yeo[8] · Jin Zhang[7]

Received: 23 June 2021 / Accepted: 13 December 2021

Abstract
The reconstruction of the trajectories of charged particles, or track reconstruction, is a key computational challenge for particle and nuclear physics experiments. While the tuning of track reconstruction algorithms can depend strongly on details of the detector geometry, the algorithms currently in use by experiments share many common features. At the same time, the intense environment of the High-Luminosity LHC accelerator and other future experiments is expected to put even greater computational stress on track reconstruction software, motivating the development of more performant algorithms. We present here A Common Tracking Software (ACTS) toolkit, which draws on the experience with track reconstruction algorithms in the ATLAS experiment and presents them in an experiment-independent and framework-independent toolkit. It provides a set of high-level track reconstruction tools which are agnostic to the details of the detection technologies and magnetic field configuration and tested for strict thread-safety to support multi-threaded event processing. We discuss the conceptual design and technical implementation of ACTS, selected applications and performance of ACTS, and the lessons learned.

Keywords Track reconstruction · Software · Pattern recognition · Vertex reconstruction · Collider physics · Concurrent event reconstruction

✉ Andreas Salzburger
andreas.salzburger@cern.ch

1 Eötvös Loránd University, Budapest 1053, Hungary

2 Institut für Physik, Johannes Gutenberg-Universität Mainz, 55128 Mainz, Germany

3 Deutsches Elektronen Synchrotron, 22607 Hamburg, Germany

4 Stanford University, Stanford, CA 94305, USA

5 CERN, 1211 Geneva, Switzerland

6 Université Paris-Saclay, CNRS/IN2P3, IJCLab, 91405 Orsay, France

7 Institute of High Energy Physics, Chinese Academy of Sciences, Beijing 100 039, China

8 Department of Physics, University of California, Berkeley, CA 94720, USA

9 Oak Ridge National Laboratory, Oak Ridge, TN 37831, USA

10 University of Massachussets, Amherst, MA 01003, USA

11 Universität Bonn, 53012 Bonn, Germany

12 Universität Hamburg, FB Informatik, 20148 Hamburg, Germany

13 Georg-August-Universität Göttingen, 37073 Göttingen, Germany

14 Lawrence Berkeley National Laboratory, Berkeley, CA 94720, USA

15 Universität Regensburg, 93053 Regensburg, Germany

16 Technische Universität München, 80333 München, Germany

Published online: 13 April 2022

Introduction

Track reconstruction will become the most computationally intensive component of event reconstruction, because it scales combinatorially with increasing number of charged particles. At proton–proton (pp) colliders such as the Large Hadron Collider (LHC), the increasing multiplicity is usually due to an increase in the simultaneous pp interactions per event, or pile-up (μ). For heavy-ion collisions, on the other hand, the particle multiplicity is primarily determined by the centrality of the event, which depends on the number of nucleon participants in each collision. For most tracking algorithms, the execution time scales approximately quadratically with the charged particle multiplicity.

In the general-purpose detector at the LHC, ATLAS [1], for example, there are currently an average of approximately 500 charged particles with sufficient momentum to be reconstructed within the detector acceptance. However, the upgrade of the LHC, the High-Luminosity LHC (HL-LHC) [2], which is expected to begin data-taking in 2027 will increase the instantaneous luminosity by a factor of five. The higher luminosity will result in an increase of the pile-up from the current average of 34 to 140–200 in ATLAS and the second general-purpose detector at the LHC, CMS [3]. The acceptance of the upgraded detectors will approximately double and additional detector layers will be added increasing the number of read-out channels. This means that there will be an average of 4000 charged particles within the upgraded detector acceptance and current minimum momentum requirements [1]. The rates at which the detectors are read-out will increase by an order of magnitude. In total, there are expected to be approximately 300,000 individual detector measurements in each event. Furthermore, additional funding for computing resources is expected to be limited in the HL-LHC era [4, 5]. Figure 1 shows that the CPU resources needed for event reconstruction are expected to exceed the available computing budget by at least a factor of two. Future pp colliders, such as the hadron–hadron option for the Future Circular Collider (FCC-hh), are anticipated to have an even larger number of up to 1000 simultaneous pp collisions [6].

Future collider-based nuclear physics experiments will accumulate several thousands of charged particles from heavy-ion collisions that occur both in the nominal interaction region and farther down the beam pipe. This leads to high occupancy and also out-of-time pile-up that creates a challenging track reconstruction environment, similar to expectations for the HL-LHC.

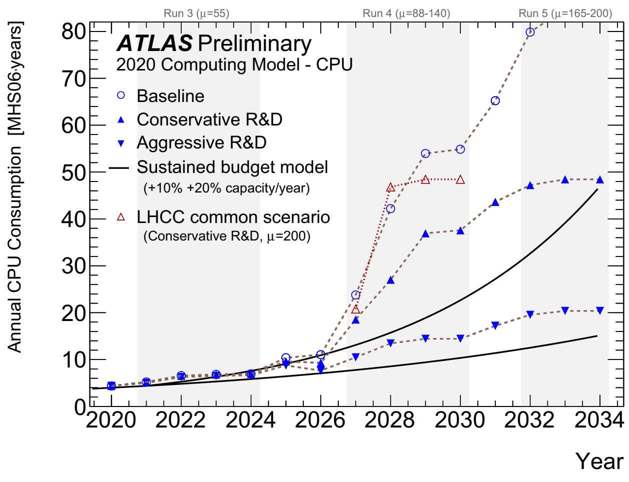


Fig. 1 Estimated CPU resources (in MHS06 [7]) needed for the 2020–2032 time frame for both data and simulation processing for the ATLAS experiment. Three different scenarios considered by ATLAS are shown ranging from the baseline to that in which the aggressive R&D program is successful (blue points). The common scenario agreed between the different experiments as a reference is shown with red triangles. The black lines indicate the amount of CPU that can be expected based on current budget models. From Ref. [5]

Historically, particle and nuclear physics have relied on Moore's Law [8], which is the observation that the number of transistors on an integrated circuit approximately doubles every 2 years. However, in the last decade, the current processor technologies have become limited in terms of the clock speeds that can be obtained due to the power density. Therefore, recent increases in speed have been achieved by adding processing cores instead of increasing the speed of individual cores. Further throughput increases are expected to be achieved through the use of different computing architectures such as Graphics Processing Units (GPUs), Field Programmable Gate Arrays (FPGAs), or integrated System on a Chip (SoC) circuits. See Ref. [9] for a recent discussion about the evolution of these technologies. Exploiting these architectures demands increasingly parallelized code and changes to programming paradigms. In addition, the rapid advances in the fields of artificial intelligence and machine learning have resulted in a wide range of new ideas for tracking algorithms. These include cellular automata [10], graph neural networks [11], and similarity hashing [12] amongst many others. While no algorithm has yet emerged to displace existing track reconstruction methods, it is still early in the development cycle for such algorithms and the field of machine learning is undergoing rapid evolution.

During event processing, the raw signals from the detectors are processed to obtain the reconstructed objects used for physics analysis. Using information from dedicated

[1] As the number of charged particles decreases rapidly with transverse momentum, the transverse momentum requirement can be raised to decrease the CPU time of track reconstruction algorithms. However, this must be balanced by its impact on the physics program of the experiment.

tracking detectors, sophisticated algorithms are used to reconstruct the trajectories of charged particles from the energy they deposit in the detector elements, including solid-state detectors with segmented read-out, gas tubes, or other tracking devices. Such track reconstruction algorithms can be considered to be part of a more general class of pattern recognition algorithms. Track reconstruction algorithms have been used in particle and nuclear physics experiments for more than half a century.

Track reconstruction methods [13] can be categorized as global and local methods, although the two categories cannot always be strictly separated. Global methods find trajectories using the entire detector's measurement ensemble, often through conformal mapping or transform methods, such as the Hough transform [14, 15]. Other global approaches use neural networks [16] to find connected sets of measurements. Local methods generate track seeds and search for additional hits to complete them. Local methods include the track road and track following methods such as the Kalman filter (KF) [17–19].

A Common Tracking Software (ACTS) entered this rapidly evolving ecosystem in 2016. It began with a small team at CERN and has since grown into an international collaboration with approximately 15 regular contributors. ACTS has its origins in the track reconstruction algorithms developed for and extensively used by the ATLAS experiment [20]. ACTS is an attempt to develop community-driven track reconstruction software, where community contributions and extensions are explicitly encouraged. ACTS provides algorithms for track reconstruction within a generic, framework- and experiment-independent open-source software toolkit [21–23]. ACTS includes data structures and algorithms for performing track reconstruction in addition to a tool for fast track simulation. The ACTS code is designed to be inherently thread-safe to support parallel code execution, and data structures are optimized for vectorization, which will speed up linear algebra operations. The implementation is designed to be fully agnostic to detection technologies, detector design, and the event processing framework to allow it to be used by a range of experiments. However, tuning of the algorithms for specific detectors is required to achieve the ultimate physics performance. Experiment-specific adaptions and tuning of the toolkit, including contextual data such as detector conditions and alignment, are made possible in ACTS through C++ compile-time specializations. In addition, ACTS is designed to be highly customizable and extendable to provide an R&D platform for the development and study of novel algorithms and techniques.

An early version of ACTS has been used to simulate the dataset for the Tracking Machine Learning (TrackML) challenge [24–26], which was performed in two stages to invite collaborators from within and external to particle physics to stimulate the development of new ideas for track reconstruction. The dataset produced for this challenge has subsequently been used to explore a range of novel track reconstruction algorithms [12, 27–30]. We use this dataset to demonstrate the current performance of the ACTS algorithms, although no rigorous performance tuning has been done. This document describes the concepts, design, and implementation of the ACTS toolkit, and does not attempt to quantify its ultimate performance on any specific detector setup. ACTS has been explored for a range of different detectors including Belle II [31], CEPC [32, 33], sPHENIX [34–36], PANDA [37, 38], FASER [39], and the future ATLAS Inner Tracking system (ITk) [40–42] for the HL-LHC data-taking era.

The concepts, design, and implementation of the ACTS project are presented here. For further details of the implementation, see the current release, Ref. [43]. "Conceptual Design" discusses the concepts and design of the ACTS software. The technical implementation is discussed in "Technical Implementation". Selected applications and early performance studies of tracking and vertexing are discussed in "Applications and Performance". "Experience" highlights some lessons learned from the experience. The conclusion and a brief outlook are covered in "Conclusion".

Conceptual Design

The ACTS project was initiated to serve three primary goals. First, to preserve and advance the well-tested code bases from the LHC experiments, while enabling preparation for the HL-LHC era and other future particle and nuclear physics experiments. This requires a state-of-the-art software development environment that allows the contributors to work with modern programming language standards and development workflows. Second, to provide an R&D test bed for algorithmic research (including machine learning techniques) and portability to accelerated hardware. Third, to ultimately provide a mature track reconstruction toolkit, that can be used as a platform for rapid development of tracking applications for future tracking detectors.

Software development for particle and nuclear physics experiments is subject to a number of constraints: an event processing framework steers the execution of algorithmic blocks, and a well-defined event data model (EDM) holds the event information and defines the communication between different components. Examples of event processing frameworks include Gaudi [44] used by the LHCb experiment [45], the Athena [46] extension of Gaudi for the ATLAS experiment, CMSSW [47] for the CMS experiment, and the ROOT [48] event processing loop. In recent years, many of these processing frameworks have been adapted and extended to enable multi-processing or multi-threaded workflows to accommodate different types of hardware

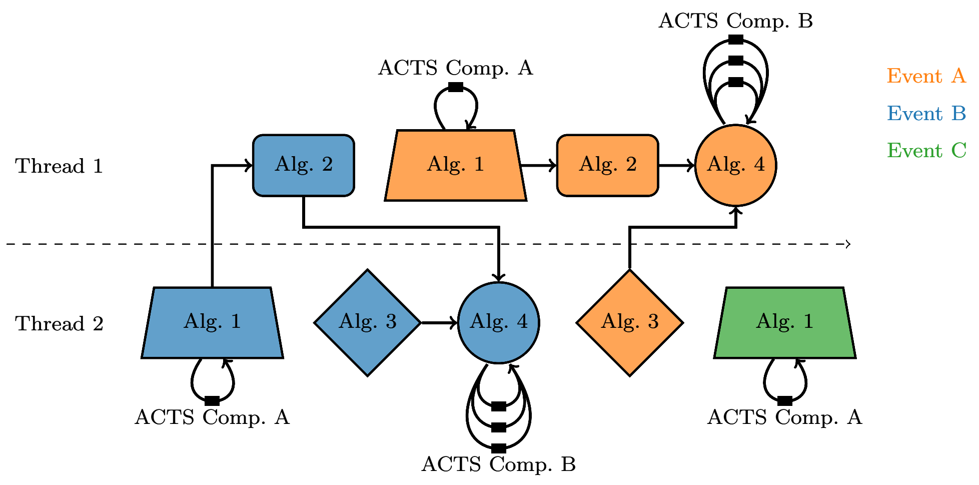


Fig. 2 Illustration of multi-threaded event processing with the sequence proceeding from left to right, in the context of an experimental software framework. Two threads execute different experiment-specific algorithms, which are illustrated by different shapes. The algorithms are distributed across threads by a scheduler. Execution occurs out of order for the three events indicated by different colors. Data flow integrity, drawn as arrows connecting algorithms, is respected. ACTS components can be used inside the algorithms, shown as loops attached to individual algorithms instances. They can optionally increase concurrency by running on parts of the event simultaneously, as shown for algorithm 4

and optimize the usage of memory and computing cores. The details of the implementation of such workflows differ between the various frameworks and experiments, but the overall concepts are the same. If a single data slice, traditionally called an event, needs to be processed by multiple threads, the function calls need to be independent of the current data slice or be provided with the appropriate context, as discussed in "Concurrent Code Execution". In this case, the method call is fully controlled and defined by the input and output data, and the algorithmic module is a stateless engine that has no memory of previous calls, configurations, and operations. Despite the complex steering and brokering of the processing, the actual work load is performed by smaller modules or tools, which are not necessarily controlled by the framework's public interface. ACTS aims to provide such a toolkit for track and vertex reconstruction, together with a high-level EDM definition that can be directly included in experiment-specific applications, extended by adding additional functionality, and rearranged and adapted to the specific needs of an experiment.

To prepare the ACTS toolkit for such general use, its design has the following central concepts:

- Minimal dependency of the core components on external software packages
- Abstraction of the EDM and geometry description from the specific details of any experiment
- General mathematical formulations of algorithms independent of specific detector geometry, magnetic field, or detector technology
- A customizable connection to the algorithm configuration
- Transparent import and handling of experiment-specific contextual conditional data, such as detector calibration and detector alignment
- Facilitation of the integration of core functionality typically governed by the event processing framework, e.g., message logging
- A plugin mechanism for extending the toolkit with external software packages.

Several of the key concepts of the design of ACTS are described in further detail in the following. The implementation is discussed in "Technical Implementation".

Concurrent Code Execution

ACTS is designed to accommodate the heterogeneous computing landscape with parallel code execution paths. Therefore, all algorithmic modules can be called in parallel while processing an event and between the processing of multiple events without interference, as illustrated in Fig. 2. The contextual and conditional data are handled transparently as described in "Contextual Data Handling". To avoid restricting the caller code to any predefined pattern, all ACTS modules are designed, such that each function call has to be fully controlled by the data input and output flow, and back channel communication to caller functions is forbidden[2]. If

[2] In C++, this is enforced by restricting methods to follow a const-correct signature and by forbidding mutable data members.

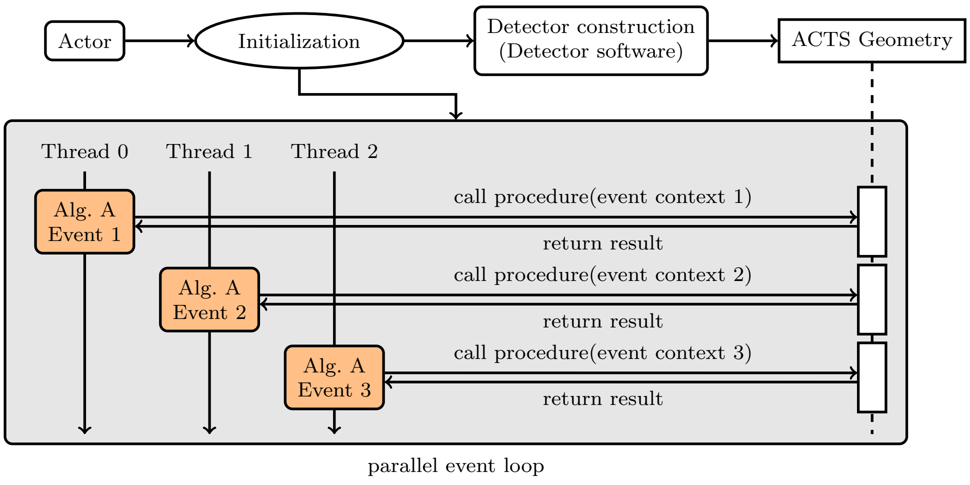


Fig. 3 Illustration of contextual geometry handling. At job initialization time, only a nominal (or initially aligned) version of the ACTS geometry is built. Three threads execute on events in parallel. All threads request details of the ACTS geometry by providing their event context, which fully defines the alignment of the detector in the current call context. The method to perform the alignment can be experiment-specific

caching is required, e.g., for performance reasons, the cache must be provided as part of the input data, as discussed in "Technical Implementation". The correct and reproducible behavior of the code in sequential and concurrent code execution paths is tested within unit and integration test suites. These tests include checks for identical results when running in single-threaded and multi-threaded mode. More advanced examples test the correct behavior with multiple alignment or magnetic field conditions during a single execution run.

The actual code execution pattern, e.g., event parallelism, or intra-event parallelism at different stages of track reconstruction, is the responsibility of the caller application and thus, no technology, language, nor dedicated library for parallel code execution is provided in the ACTS core modules. However, example applications in the repository rely on the Intel Threading Building Blocks (TBB) [49] multi-threading library to demonstrate how concurrent execution can be implemented. ACTS allows core modules to be wrapped in callable kernel structures that can be used on accelerators with dedicated technology back ends. First demonstrators of such an approach have been successfully deployed [50]; however, further development and simplification of the code base is needed for ACTS to run efficiently on different types of computing hardware. This is one of the dedicated R&D lines of the project as discussed in "Research and Development Projects".

Contextual Data Handling

A general track reconstruction toolkit that serves different experiments must be able to handle a contextual experimental environment. Detectors may have temporary or permanent imperfections, suffer from changing alignment and data-taking conditions, and, in general, operate in a time-dependent manner. Track reconstruction uses high-precision measurements and every effect must be accounted for to achieve optimal results. Detector conditions, on the other hand, are one of the most specific aspects of any experimental setup, and a general solution or implementation for such a diverse problem would be very challenging. Therefore, a transparent handling schema for all contextual data has been applied throughout the ACTS code base: a set of contextual objects, defined and implemented in the experiment's software stack, are handed through the entire call structure of ACTS (see Fig. 3). This ensures that each geometry call that relies on detector information is aware of the geometry context of that particular call and allows the correct detector alignment to be applied within that specific call context. Other conditional data, such as the magnetic field status or detector calibration data, are implemented in the same way. In all cases, the caller code can be assured that contextuality will be respected with minimal computational overhead, because the context information is unpacked and correctly interpreted. The choice about whether the contextual object carries either a parameter to identify the context to be applied or the full contextual data is left to the implementation within a particular experiment.

See "Selected Applications" for details of a concrete implementation of a contextual environment.

Research and Development projects

Recent technology advances in both hardware and software have transformed the computing landscape in the scientific and private sector. Machine learning is a rapidly growing field, and hardware-based acceleration becomes increasingly prominent due to the growing use of high-performance computing centers and limitations in increases in processor speed. While both areas have already been explored in particle and nuclear physics, additional R&D is needed to fully exploit these advances in future data processing, particularly in the domain of track and vertex reconstruction. The tracking machine learning challenge has demonstrated that machine learning algorithms can reach the same order of magnitude in both physics performance and execution speed compared to the current track reconstruction algorithms. End-to-end solutions based on machine learning are expected to require significant development time. However, certain aspects of track reconstruction such as track classification or data segmentation [51], have already shown promising results. Such smaller components, however, need to be tested in a realistic data flow. A key element in the design of ACTS is to provide a playground to facilitate prototyping, development, and testing of such new ideas. The plugin mechanism of ACTS allows the core track reconstruction code to be coupled with external libraries from the machine learning and data science sectors, or with code with different language backends, which is needed for code execution on accelerators. The `ONNX` [52] library for the deployment of machine learning-based tracking solutions and the `autodiff` [53] library for automated compiler-based differentiation have both been demonstrated within ACTS. Furthermore, `CUDA` [54] and `SYCL` [55] have been integrated for GPU-based seed finding algorithms.

Technical Implementation

Basic Technology Choices

ACTS targets modern many-core, general-purpose CPUs, which are widely available and the default computing architecture currently used by the LHC experiments [56, 57] and other experiments in particle and nuclear physics. Both x86 and ARM architectures have been demonstrated to work with ACTS. All recent CPUs have vector units and significant performance improvements can be obtained from vectorizable code. While hardware accelerators such as GPUs and FPGAs are not necessarily part of most baseline architectures, they are actively explored by the ACTS developers and the larger particle and nuclear physics community, particularly for online software and when looking ahead towards the HL-LHC.

ACTS is implemented in C++ 17 [58], which is widely used in the particle and nuclear physics community, and thus can be easily integrated with existing software. As a compiled programming language with minimal implicit runtime facilities and a high degree of low-level hardware control, C++ enables achieving excellent execution performance. However, it is difficult to learn and use correctly, particularly with regards to memory management. This is mitigated through guidelines and implementation choices in ACTS, which include strict ownership handling via movable types and value-like semantics as well as the adoption of best practices such as unit tests and continuous integration, which ensure code quality.

The ACTS code is designed to have minimal dependencies on external packages. Only two third-party libraries are required: `Eigen` [59] for linear algebra and `Boost` [60] for unit testing, file system handling, and a few key containers.

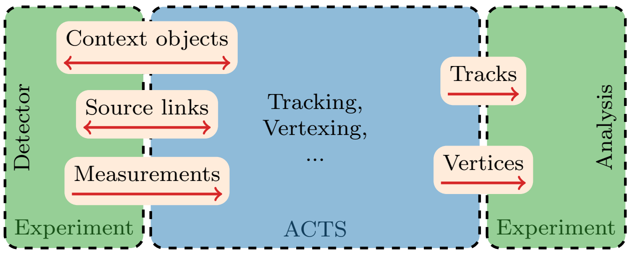


Fig. 4 Example of the integration of ACTS into an experiment's software framework. The experiment- and detector-specific code (green) is expected to handle low-level data preparation and provide `SourceLinks` and `Measurements` as input to ACTS algorithms. ACTS provides tracks and vertices as output for further experiment-specific reconstruction and analysis

In addition, `CMake` [61] is used both as a dependency management tool and as the build system.

The general strategy for algorithmic development in ACTS draws on the experience of previous particle and nuclear physics software efforts and, in particular, the existing ATLAS offline tracking software [62]. A key choice is to favor small compile-time interfaces and templates over virtual interfaces for better performance and greater implementation flexibility. ACTS favors data-oriented programming over object-oriented programming, which means that the communication between different parts of the code occurs through sharing common data structures rather than predefined interfaces.

Code Organization

The ACTS code [43] is a single open-source repository hosted on GitHub [63]. A single repository allows for the easy development and integration of components and avoids version mismatches and accidental incompatibilities. The test and validation code provides examples of end-to-end tests and allows development with realistic reconstruction chains. No additional client application is necessary. The use of GitHub facilitates collaboration independent of affiliation. Key components within the repository are the core library, the plugins, the fast tracking detector simulation, Fatras, based on the original ATLAS fast track simulation [64], and the test and validation code. Figure 4 illustrates how ACTS can be integrated into an experimental framework.

The core library implements the basic tools and key algorithms with minimal external dependencies. The plugins directory contains core-like functionality that requires additional external dependencies. The source code of the core library and the plugins extensions are located in the `Core` and `Plugins` directories. Examples available in the plugins directory include geometry tools based on the external `TGeo` [65] package from the `ROOT` toolkit, which is currently used in particle and nuclear physics experiments

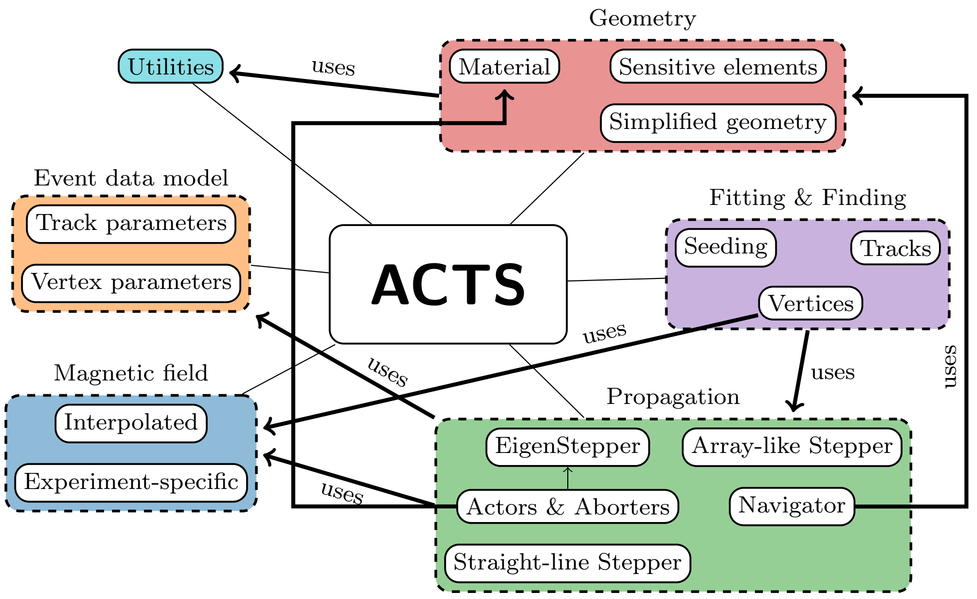


Fig. 5 Overview of selected components in the ACTS repository and their interactions. The components are categorized into modules, such as geometry, propagation, or event data. A non-exhaustive number of relationships where one component "uses" other components in different modules are indicated by arrows. The stepper components are connected to the magnetic field module, because they are used to retrieve the magnetic field information

to describe detector geometries, and CUDA or SYCL which enables code to run on GPUs.

The source code of the Fatras simulation is not located within the Core folder, because it is not required for reconstruction. It can be built on demand. Fatras makes heavy use of the core functionality and, therefore, can be maintained more easily as part of the same repository, e.g., to adapt to core interface changes.

Releases of ACTS follow semantic versioning [66], where a subset of the interface is considered when determining the major version. The software is provided under the Mozilla Public License, v. 2.0 (MPLv2) [67]. Common code formatting is ensured by requiring submitted code to the repository to pass a formatting check using the clang-format [68] LLVM [69] extension.

Core Components

The core library of ACTS is organized into modules and each module groups tools and algorithms with similar functionality. An overview of key modules is shown in Fig. 5. The communication between algorithms occurs via common event data structures defined in the EventData module as described in "Event Data Model". The Geometry module handles the tracking geometry, which is the logical and geometric grouping of detector surfaces into layers and volumes. The tracking geometry uses the Surfaces component, which implements different surfaces for detectors and boundaries. The related Material component contains tools to describe surface- and volume-based material and the algorithms to create such a geometrical mapping. See "Geometry" for further details about both modules.

The Propagator module provides tools to propagate particle states along their trajectories in different magnetic fields (see "Propagator"). The TrackFinding and TrackFitting modules use both the Geometry and the Propagator modules. The Vertexing module is largely standalone, but relies on output from other modules as input and the propagation infrastructure. The Seeding module contains a geometry independent seeding algorithm that acts purely on global three-dimensional points.

Configuration, State, and Context

```cpp
class Algorithm {
public:
    // Necessary and sufficient
    // configuration variables.
    struct Config {
        double aValue = 0.25;
        std::string name = "something";
    };

    // Construct algorithm from its configuration.
    Algorithm(const Config& cfg) : m_cfg(cfg) {}

private:
    Config m_cfg;
    // ... e.g. values derived from
    // the configuration
};
```

Listing 1: Example of an algorithm implementation with a nested configuration type. The configuration type allows settings with default values to be addressed by name. Plain types mean that no explicit Config constructor is required. This pattern is used throughout the code base to support configuration and construction. This listing also demonstrates the naming convention for private member variables and the CamelCase coding style.

```cpp
class StatefulAlgorithm {
public:
    // Cached or intermediate values that
    // persist between calls.
    struct State {
        double lastValue = 0.;
    };

    // `const` implementation since all mutable
    // states are contained within the state object.
    void doWork(State& state) const;
};
```

Listing 2: Example of a stateful algorithm implementation with a nested state type. All state information, i.e. either cached or intermediate values, are contained in the nested state type. The algorithm type itself remains stateless and const-correct.

Components in ACTS are typically highly configurable. To enable this flexibility, without being bound to any specific configuration environment, patterns using a nested C++ structure are used. Listing 1 provides an example of such a pattern, where a `Config` structure contains all configuration parameters as members. The constructor of the outer type takes an instance of the configuration structure as an argument, and runs its setup accordingly.

ACTS supports both inter- and intra-event parallelization without expliciting implementing either. Instead, the explicit state objects for potentially stateful algorithms must be provided by the user as demonstrated in listing 2. An example of a stateful algorithm would be, e.g., a track finding algorithm that uses information about previously found tracks in the event (provided by the state) to prevent unnecessary or duplicated track search. By creating these state objects within their own framework, experiments must explicitly decide how and at which levels execution is parallelized and where synchronization might need to occur.

A similar problem exists for detector-related structures including the geometry, magnetic field, or calibrations that vary between events. During parallel execution, these structures cannot be handled as global states. Similar to the handling of the algorithm state, all algorithms that might require varying context data take an explicit context object. These objects are then passed through the full execution chain and handled by the experiment-specific code where necessary. An example of an application with contextual data, changing the detector alignment, is demonstrated in "CPU Utilization".

Event Data Model

The event data model binds all modules together by providing shared data structures. The EDM is used to communicate between different steps of the reconstruction chain. Thus, it needs to be both generic enough to hold all possible event data types, but also minimal enough to avoid overheads, as it will be used extensively throughout the code base. Event data consists of measurements, track parameters, and vertex parameters, which can be represented as vectors.

The two different track parameter spaces in ACTS are bound and free track parameters. Bound track parameters describe a track bound to a surface. The surface can be a real detector surface such as the planar surface of a silicon detector or a virtual surface, such as the straw surface and the perigee [3] surface used to describe a track near an anode wire in a gaseous tracking detector and a vertex, respectively. The bound track parameters have six dimensions and comprise of a two-dimensional position on the local surface, two momentum direction angles (or angle-like parameters), a curvature parameter, and time. The bound parameters can only be defined with reference to a surface and the interpretation of the two local position components are surface-dependent. At the perigee surface, the bound track parameters are

$$\mathbf{b} = \left(d_0, z_0, \phi, \theta, \frac{q}{p}, t\right), \tag{1}$$

where the d_0 and z_0 represent the transverse and longitudinal impact parameters, respectively. The remaining parameters are the azimuthal angle ϕ, the polar angle θ, the charged signed inverse momentum, and the time t. This parameterization exists for charged and neutral particles. In the latter case, the inverse momentum representation is changed to $1/p$. The time t is transparently respected in track propagation and potential measurement inclusion.

In contrast, free parameters require no reference surface and use the same definition everywhere. Within ACTS, they are described by the 3D position and direction vectors, time, and a curvature parameter. Therefore, they are eight-dimensional and are used throughout track propagation

$$\mathbf{f} = \left(x, y, z, d_x, d_y, d_z, \frac{q}{p}, t\right), \tag{2}$$

and also referred to as free track parameters.

Measurements are treated as vectors in a sub-space of a (bound) track parameter vector space. Measurements are typically associated with a surface and only measure a subset of the available track parameters; most often at least one

[3] The perigee refers to the point or surface of closest approach.

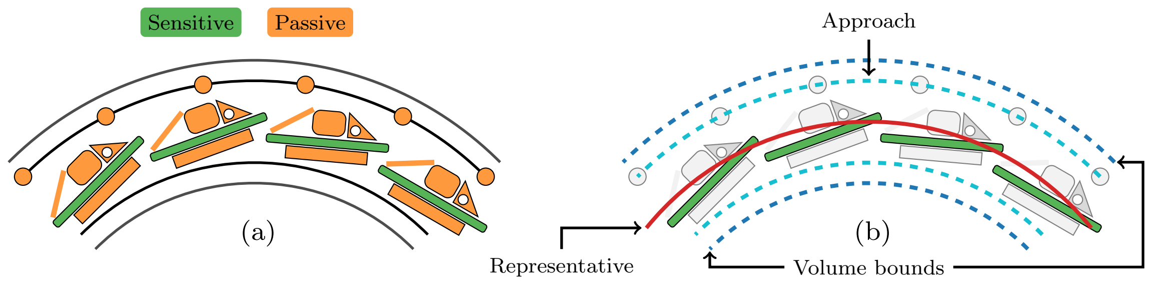


Fig. 6 Illustration of the layer geometry for planar detection modules. **a** Highly detailed geometry, in which both sensitive and passive elements are present. **b** Simplified version, where all passive elements are discarded (grayed out). Instead, various virtual surface approximations of the detailed structure are shown and used in the modeling. The representative surface is closest to the sensor locations, while the approach surfaces form an envelope around them. A volume surrounds the layer, which also features boundary surfaces

local position. Many track reconstruction methods, such as the Kalman filter (see "Kalman Filter"), include a projection from a subset of the track parameters to the measurement space. For ACTS, the measurement space is assumed to always be consistent with the bound track parameter space defined by the surface. The inclusion of time information directly in the track fit is a novel feature of the ACTS algorithms. For example, a pixel detector measurement with time information $\mathbf{m}_i = (m_x, m_y, m_t)$ can be compared to the estimated track parameterization $\mathbf{b}_i = (l_x, l_y, \phi, \theta, \frac{q}{p}, t)$ on the same surface i using a projection matrix $\mathbf{H}_i$ to form a three dimensional residual vector: $\mathbf{r}_i = \mathbf{m}_i - \mathbf{H}_i\mathbf{b}_i$. Time is treated in the same way as the other track parameters.

Compile-time programming via template substitutions is used to dispatch execution into highly optimized code paths for each dimensional measurement type. A separate data-structure provides an optimized collection of measurements. This data-structure can also store a tree of *track states*, each potentially containing a measurement and/or the estimated track parameters.

The dedicated event data model used by the vertexing components is designed to be as flexible as possible and the input tracks can be of any user-defined type. This approach facilitates experiment-specific integration while keeping overhead minimal at the same time.

Geometry

The geometry description used for reconstruction is a simplified version of the detailed detector description used in Monte Carlo simulation programs such as `Geant4` [70]. The description of the sensitive detectors (including misalignment and other contextual information) needs to be as precise as possible. However, several approximations to the detector description for the non-sensitive detector elements are made. During reconstruction, the noise from the detector material is accounted for either deterministically or stochastically.

In ACTS, the reconstruction geometry is entirely built from surface objects. Compound layer objects and volume objects are based on the surface class. A volume shape is built from the boundary surfaces. The boundary surfaces are also referred to as portal surfaces as they connect the volumes. Layers are defined by their bounding and contained surfaces. The contained surfaces can either be declared *sensitive* when they represent detection elements or be passive material surfaces.

Navigation through the detector proceeds either using portal surfaces that connect volumes with other volumes or by performing a local search of layer surfaces after entering a layer object through its bounding surfaces. All surfaces can be propagated to, carry material, or refer to sensitive detector elements, and are thus suitable for both reconstruction and fast simulation.

Layer Geometry and Plugin Mechanism for Detector Elements

Tracking detectors are frequently built from physical layer structures that support the modules, the on-detector electronics, power cabling and cooling units, and often feature stave structures. The logical division into layer structures is used in ACTS to restrict the local navigation to an area of interest instead of attempting to navigate the full detector.

Each layer has a set of approach surfaces, as well as a representative surface, which is a single surface representing the layer in a fast navigation search. The approach surfaces describe the boundary of the layer and are the entry point into the local layer navigation. In track propagation, the intersection of the approach surface is used for finding possible surface candidates within the layer object that are then tested for intersection with the trajectory. The different types of surfaces are illustrated in Fig. 6.

ACTS allows this generic geometry description to be supplemented with experiment-specific information. Each sensitive surface can have an associated object containing specific information of the particular experiment. For example, this can be used to interface with an experiment's geometry library. In addition, ACTS ships with plugins which can be used to translate a geometry from an existing representation, such as `DD4Hep` [71], `TGeo`, or `GeoModel` [72].

Surface and Volume-Based Material

In addition to defining the exact positions and shapes of the measurement devices, the detector geometry description must provide an adequate description of the detector material. Because passive and active material is the main source of uncertainty in track reconstruction, a precise description of the amount, type, and location of the material in the detector volume is required. The passive material can be handled as either deterministic changes to the trajectory estimate or stochastic addition to the covariance matrix.

While a precise description is required for the simulation of individual interactions of the particles with the detector material, it can be simplified for track reconstruction. The material can usually be approximated as average material mixtures, described by an effective amount of traversed radiation length for evaluating the multiple scattering and bremsstrahlung contributions, and an effective ionization loss can be applied. Furthermore, small structures present in the full simulation geometry can be merged into close-by approximate material slabs. This simplification speeds up the track reconstruction algorithms, because navigating and propagating through a simplified geometry require fewer CPU cycles, predominantly due to the reduction of surface candidate intersections and fewer calls to the material integration calculations. The relative importance of accuracy and speed must be optimized for each experimental setup. ACTS deploys a highly configurable approach to this problem: every surface and every volume can carry an attached material description, including the auxiliary layer surfaces and volume boundary surfaces. Depending on the environment, corrections need to be applied during track propagation as described in "Propagator", which require a precise description of the material. Therefore, the dedicated mapping algorithm in ACTS projects the detailed material description onto a selected set of surfaces or into a selected set of volumes. An example of a material mapping application is discussed in "Applications and Performance". The material description on surfaces or within volumes can be either homogeneous or binned, using the ACTS grid infrastructure. When the propagation reaches a surface that carries material, the appropriate material integration methods will be called. Similarly, if the propagation proceeds within a volume that carries a material description, the corresponding extension for the transport equations become active and query the volume material.

Propagator

A core module of ACTS is the propagation engine, which carries out the task of transporting track parameters through the detector. Minimum requirements for the propagator include reliable navigation through all the detector components and the mathematical transportation of the track parameters and their associated covariance matrices. Additional actions can be performed during track parameter transport in both track reconstruction and fast simulation. An example of such an action is the intersection with additional sensitive modules. This can be used to count the number of missed sensitive detector element on a track, log several parameters, or execute any particular action that can be performed along a particle's trajectory. The propagation engine therefore consists of two components:

1. A *Stepper* module which performs the mathematical transport through the magnetic field
2. A *Navigator* module which predicts the potential candidate surfaces in the detector geometry and regulates the associated step size for the stepper.

The propagator is steered by a dedicated options object, which is provided for each propagation call. It contains two

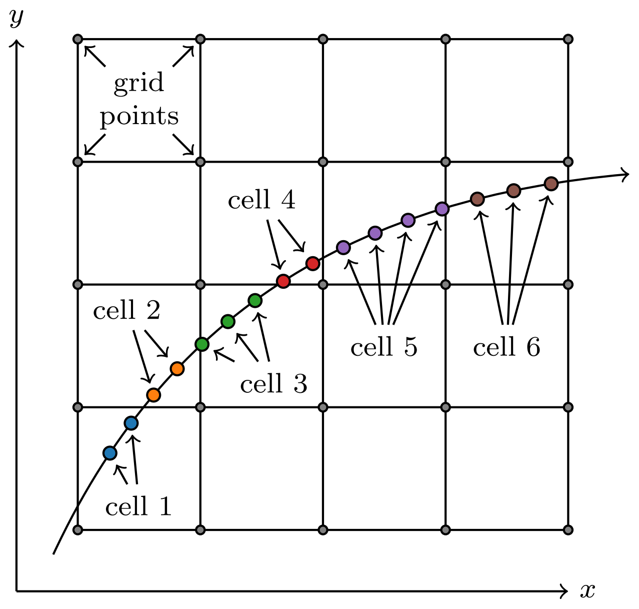


Fig. 7 Illustration of the magnetic field cell implementation. A two-dimensional field map in the xy-plane is shown. The colored circles represent propagation steps where magnetic field lookup is performed. Step locations that fall inside each lookup field cell are indicated with the same color. Before crossing the boundary into the next cell, each step reuses the previously retrieved field cell

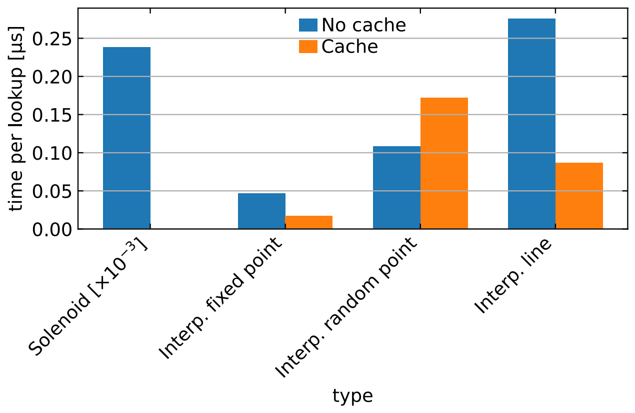


Fig. 8 Performance of the magnetic field lookup for a number of different scenarios. Results for the analytical solenoid field and the interpolated magnetic field map are shown. Field queries at a fixed point, at a sequence of random points, and a sequence along a straight line are measured. Performance with and without field interpolation cell caching is shown

lists of structures: a list of *actors* and a list of *aborters*. Both lists can be extended by the client code at compile time and are called after each propagation step and can contain surface material interaction logic (as part of the *actors*), target conditions, or restrictions on the maximum allowed path length.

```
// Pre-Stepping: target setting
navigator.target(state);

// Propagation loop: stepping
while (/* step */){
    // Perform a step & check the result
    stepper.step(state);
    navigator.status(state);
    // Apply the actors
    actionList(state, result);
    // Check for abort condition
    if (abortList(result, state)) break;
    // Target after stepping
    navigator.target(state);
}
```

Listing 3: Simplified listing of the propagation loop showing the interplay of the *Navigator*, the *Stepper*, the *actors* and the *aborters*. The `state` object holds the cached track parameterization, while the *actors* collect information in the `result` object, which in turn can be interpreted together with the `state` object for eventual abort conditions.

ACTS includes two steppers based on a fourth-order Runge–Kutta–Nyström algorithm [73]. One has an array-like math implementation and the other is based on the `Eigen` math library. These steppers receive the magnetic field as an input. For the `Eigen`-based stepper, an extension for propagation through non-vacuum material based on the simultaneous track and error propagation (STEP) algorithm [74] exists and is invoked in presence of a volume material description. A straight-line stepper also exists, which can be used in the absence of a magnetic field. A purely helical stepper is not implemented, but both Runge–Kutta–Nyström based steppers can provide helical stepping behavior for a constant magnetic field. Listing 3 provides a simplified listing of the propagation loop showing the interplay between the different components.

Magnetic Field Access

The magnetic field is accessed via a dedicated provider that is passed to the stepper modules. The implementation of the magnetic field (both in memory and in conceptual design) can be changed and a few standard implementations are provided. An interpolated magnetic field map, which implements an internal caching mechanism, is also provided and can be used to approximate any inhomogeneous magnetic field by supplying suitable input. When following a particle through the detector, calls to the magnetic field are often made in short succession. Therefore, to optimize the lookup or potential re-use of the magnetic field information, the steppers access the field via a thread-local cache type. In the implementation of the interpolated magnetic field map, this cache contains the current field interpolation cell. A successive call to the field interface either results in a renewed interpolation if the call remains within the same field cell, or the retrieval of a neighboring field cell. The field cell concept is visualized in Fig. 7, where a particle trajectory is shown in the *xy*-plane with the propagation step locations color-coded according to their respective field cell.

Figure 8 shows the performance of the ACTS magnetic field interpolation for different scenarios. A dynamically calculated solenoidal field is shown as a baseline. From that field, an interpolation grid is derived during initialization, and its field lookup performance is measured for a number of access patterns: a fixed point, random points, and a sequence of points along a straight line. The last emulates the typical access pattern of particle propagation. All interpolated field query strategies are approximately three orders of magnitudes faster than the solenoidal field calculation. The impact of the field cell cache is also shown. For the fixed point, the caching results in significant performance improvements, while for fully random points, it degrades the performance. This is expected, because random point access will almost always result in a cache miss, while for fixed point, a cache hit is guaranteed. For the straight-line access pattern, the cache again improves performance.

Track Seed Finder

Track seed finding algorithms are used as the first step in track reconstruction to obtain a coarse estimation of the possible track candidates and their properties, which are then used by track following algorithms. The current implementation of the track seed finder in ACTS creates triplets of measurements, with the goal of identifying the triplet of measurements corresponding to a single particle. In track seed finding, the goal is to maximize the efficiency while minimizing the number of seeds that do not correspond to a particle, or *fakes*, and duplicates. Maximizing the efficiency is the highest priority, because particles without a seed will never be reconstructed as tracks, while fakes and duplicates can be eliminated in subsequent steps of the track reconstruction chain at the cost of execution time.

The ACTS track seed finding algorithm takes three-dimensional measurements from specified detector components as input, and applies selection criteria to prioritize measurements which are more likely to have originated from the same particle. These criteria must be optimized for a particular detector geometry and play an important role in determining the physics and technical performance of the track seed finding algorithm. As track parameters derived from triplets have limited resolution, approximations are used in their estimation, including a homogeneous approximation of the magnetic field. Information about the detector geometry is not required during execution, because the track seed finder relies on global measurements.

The efficiency and computational performance of the track seed finding algorithm depends on the number of measurements and the event occupancy. The higher the measurement occupancy, the higher the computational cost of following all combinatorial paths and higher the number of fakes. This can be mitigated with tighter selection criteria at the cost of lower efficiency. Moreover, higher measurement occupancy results in a higher probability that a fake measurement, instead of the real one, is assigned to the track, which leads to additional efficiency loss. As the number of detector layers increases, more measurement points are available per particle, which increases the computational requirements and the duplicate rate, but also the efficiency. The accuracy of the detector alignment also impacts efficiency.

Kalman Filter

The Kalman filter technique processes a set of discrete measurements to determine the internal state of a linear dynamical system. In particular, random perturbations can be present in both the measurements and the system. It is commonly used for navigation, but has applications in many domains including charged particle reconstruction. The Kalman filter is an excellent choice of algorithm for charged particle reconstruction, because it facilitates a straightforward treatment of the motion of charged particles in magnetic fields and the impact of the detector material on the particle trajectories including multiple scattering and energy loss.

Kalman filter algorithms can be used both for track finding and track fitting. In ACTS, the Kalman filter algorithm estimates the parameters of a track by iteratively incorporating individual measurements assigned to the track by track finding algorithms. The implementation in ACTS has the mathematical filtering and smoothing in configurable components which can be replaced at compile time. The Kalman filter class includes a propagator instance which can be configured with different detector geometries and magnetic fields. The algorithm is primarily implemented in an actor that is fed into the propagator when the track fit is executed. This actor can access the transported track parameters and their associated covariance matrices, and operate on them. It is also configurable in terms of the representation of the track parameters and measurements, and can include an *outlier* [4] identification helper and a calibrator for the calibration of measurements using predicted track parameters during the fitting. As the time parameter including correlation is included in the track parameters and their associated covariance, a time measurement, if present, will transparently be used to update the predicted time parameter. The time parameter will also be propagated along the trajectory including its variance and correlations with the remaining track parameters.

The Kalman filtering method creates a track state if the propagator reaches a surface with either material or a measurement. If a measurement is found, it is investigated by the outlier identification helper. Unless the measurement is tagged as an outlier, it is used to update the track parameters by applying the filtering procedure. A *hole* track state is created on any traversed sensitive surface that does not have a measurement on it. Material effects can be included either before or after the Kalman filtering. When all the measurements have been processed or the navigation reaches the boundary of the tracking geometry, the Kalman smoothing procedure is triggered to obtain the smoothed track parameters either using the Rauch–Tung–Striebel smoothing formalism [75] starting from the last filtered track state or using the propagator but with the navigation direction reversed.

An extension of the Kalman filter (KF), the Combinatorial Kalman filter (CKF) technique [76–78] is implemented

[4] A measurement which is not compatible with the predicted track parameters is term an outlier.

within ACTS to perform the measurement search at the same time as performing the fit. If multiple compatible measurements are found on a surface, the track propagation branches and is repeated for multiple sets of track parameters updated with each subsequent measurement. The search for compatible measurements is handled by a measurement selector, which supports custom implementation of the selection criteria. An example of the selection criteria is the maximum χ^2 for each selected measurement under the assumption of the track parameters and the maximum number of branches on a surface. Those criteria are fully customizable and configurable at different levels of detector geometry, and can be used to refine the tracking performance, e.g., the track reconstruction efficiency and the number of fake tracks.

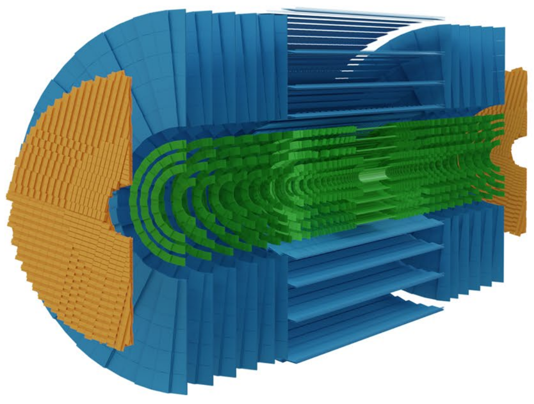
Both the KF and the CKF produce fitted track parameters at a user-defined target surface and a container object, which contains all the fitted track states. For a single seed, the KF and the CKF can provide one set and multiple sets of fitted track parameters and track states, respectively.

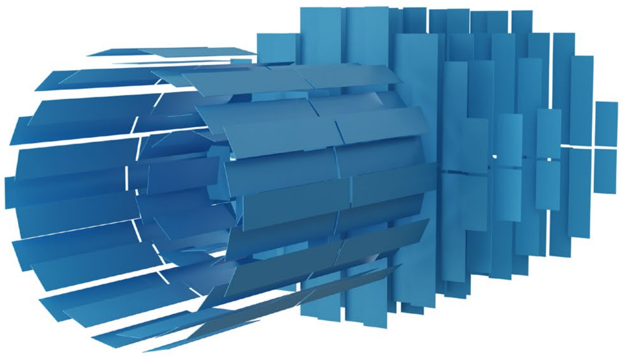
Vertexing

ACTS features a fast and flexible primary vertex reconstruction suite, comprising a range of components implementing a full chain from vertex seeding to precision vertex parameter estimation. The vertexing module includes an iterative vertex finder (IVF) and an adaptive multi-vertex finder (AMVF) [79]. The IVF iteratively fits individual vertices starting from a vertex seed and a seed track collection. The AMVF fits multiple vertices simultaneously, while dynamically assigning tracks to candidate vertices during fitting. The AMVF exhibits good performance for high vertex-density environments such as the HL-LHC, and will be used as the default vertex reconstruction tool for the ATLAS experiment in Run-3.

The input vertex seeds to both vertex finders are provided by four different vertex seed finding algorithms: a z-scan vertex seed finder based on a half-sample mode algorithm [80], a Gaussian track density vertex seed finder [79], as well as a non-adaptive and adaptive version of a new fast and robust grid density vertex seed finder. Dedicated vertex fitters for the different vertex finding approaches, a Billoir fitter [81] and an adaptive multi-vertex Kalman fitter [82], as well as auxiliary vertexing tools such as impact point estimators and track linearizers complement the vertexing toolkit.

The public interfaces of the vertexing components are designed to be highly configurable and flexible. The vertex finders accept a collection of representations of tracks


(a)


(b)

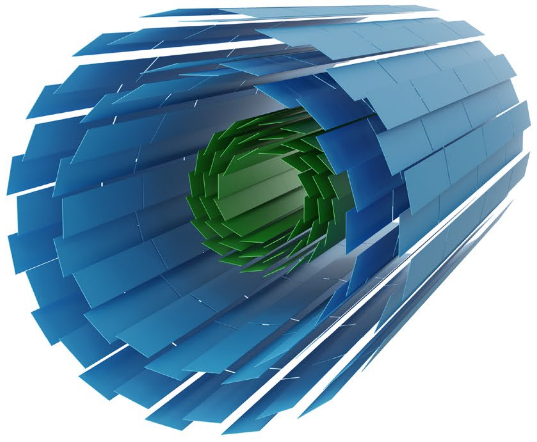
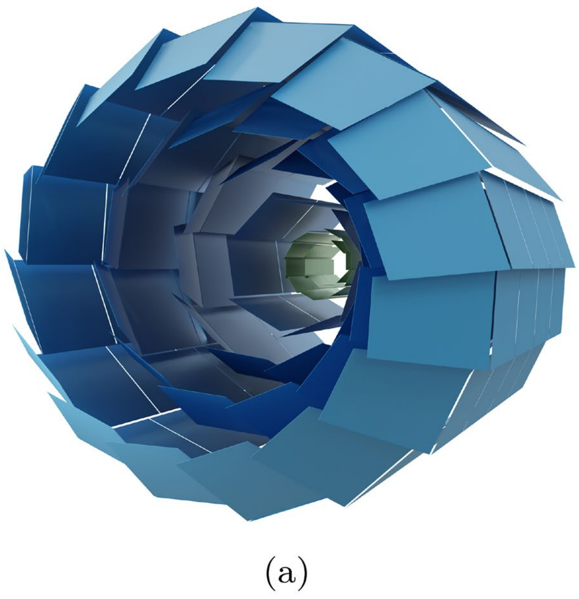

(c)

Fig. 9 The geometry of the ATLAS ITk (**a**), the PANDA silicon detector (**b**), and the sPHENIX silicon tracking detectors (**c**), implemented with ACTS. Colors indicate different subsystems; in the top image, the High Granularity Timing Detector (HGTD) [83] is shown in orange



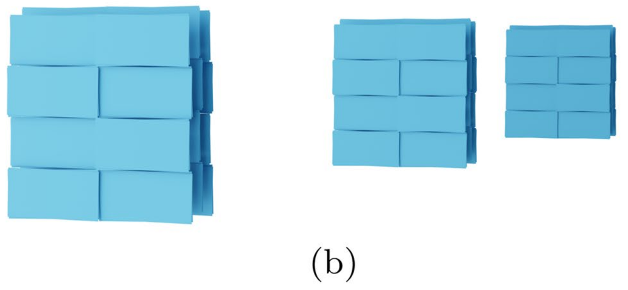
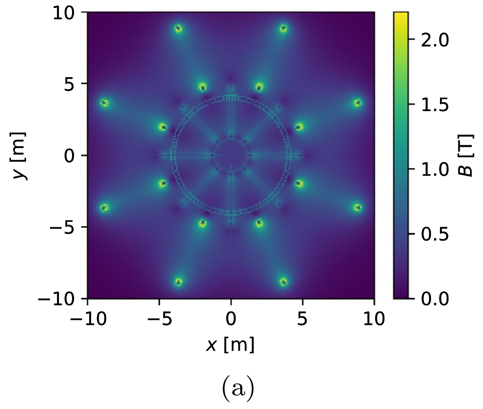


Fig. 10 Geometries of Belle II (**a**) and FASER (**b**) implemented in ACTS. Colors indicate the different subsystems



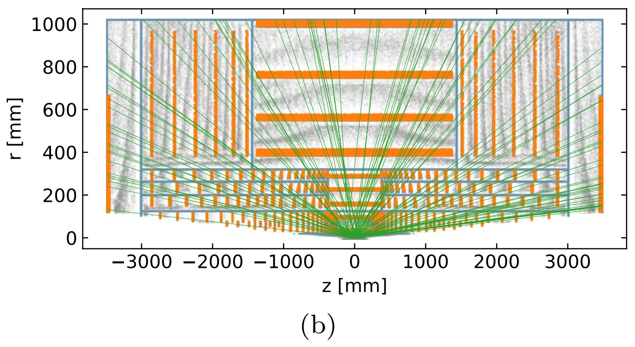


Fig. 11 A projection of the magnetic field implemented with ACTS for the ATLAS tracking system into the $x-y$ plane (**a**). The strength of the magnetic field at each point is indicated in color. The $r-z$ coordinates of the intersections of propagated pion tracks with the ATLAS ITk detector elements, using the ATLAS magnetic field (**b**). Boundary intersections are shown in blue, while intersections with sensors are shown in orange. Green lines indicate a subset of extrapolated particle tracks. Gray points are the intermediate integration steps, required within a predefined tolerance threshold

or particles to be used for vertex finding. In addition, an option structure which allows the finder to supplied with a vertex constraint is provided as input. The output of the vertexing components is a list of all found vertices. Vertex seed finders are regarded as regular vertex finders in ACTS, and therefore share the same interface. They have the special characteristic of returning a single-entry list of vertices, i.e., the vertice obtained from the current vertex seed only, at a time. The vertexing can run on ACTS bound track parameter objects as well as on any user-defined input track type to allow maximum flexibility. The only requirement for using an arbitrary input track type is to provide a `std::function` that unwraps and returns ACTS bound track parameters.

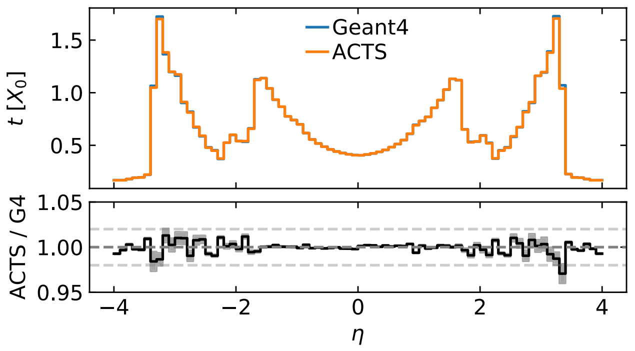


Fig. 12 Comparison of the mapped material obtained from ACTS material mapping tool (orange line) and the `Geant4` material (blue line) as a function of η for the Open Data Pixel Detector. The ratio of the material in ACTS to Geant4 is indicated in the panel below and the statistical uncertainty is indicated with the gray band. Agreement is within about 2%, with excellent agreement seen in the central part of the detector

Applications and Performance

Selected Applications

ACTS is integrated or being integrated into a number of particle and nuclear physics experiments. Here, we presented selected examples of experiments that either use or have explored the use of ACTS. Figure 9a shows the geometry of the ATLAS ITk. The ACTS vertexing algorithms have already been integrated into the ATLAS Athena framework and the integration of the ACTS tracking algorithms is ongoing. At the same time, preliminary optimization of the ACTS tracking algorithms for ITk is in progress. Figure 9b shows the geometry for the silicon tracker of the PANDA experiment, which is a planned particle physics experiment at the FAIR facility in Germany.

The sPHENIX experiment is the next-generation jet and heavy-flavor detector currently under construction at the Relativistic Heavy Ion Collider at Brookhaven National Laboratory. Figure 9c shows the geometry for the silicon tracker of sPHENIX. ACTS components for seeding, track fitting, and vertexing have been successfully deployed in the sPHENIX production software chain.

Belle II is the next-generation B-factory experiment located at the SuperKEKB accelerator complex [84] in Japan. A critical requirement is to reliably reconstruct low-momentum tracks with $p_T \approx 100 - 300\,\text{MeV}$ [85]. This is achieved with a combination of silicon pixel and strip detectors, whose placement is shown in Fig. 10a. The Belle II collaboration is currently exploring in what form ACTS can supplement or replace existing tracking code.

Figure 10b shows the FASER detector, which is an experiment at the LHC, located $\simeq 480$ m downstream the ATLAS interaction point, featuring extremely forward acceptance ($\eta > 9.2$). The FASER tracker is designed to detect two high-momentum charged tracks originating from a decay vertex inside the decay volume, using three tracking stations with silicon strip sensors, in a 0.55 T magnetic field. FASER will fully rely on ACTS for its track reconstruction and fitting. The implementation is well progressed and first performance studies with the ACTS CKF are in preparation.

Figure 11a shows the magnetic field of the ATLAS experiment described using ACTS. Track parameter propagation based on the detector geometry and magnetic field is used to determine the coordinates of intersections of tracks with detectors. An example of track propagation with the ATLAS ITk Detector is shown in Fig. 11c.

When using the simplified tracking geometry described in Fig. 3.6, the detector material is modeled using a dedicated mapping algorithm that remaps the detailed `Geant4` material. A comparison of the mapped material with the material used in the full simulation geometry for the Open Data Detector [86] is shown in Fig. 12. The geometry of the Open Data Detector is described with a realistic passive material model based on DD4hep, which translates into a `Geant4` detector model. The agreement between the material budget described in `Geant4` and by the ACTS geometry is within a few percent, and can be further improved using higher granular binning of the material maps and additional placement of material surfaces if needed.

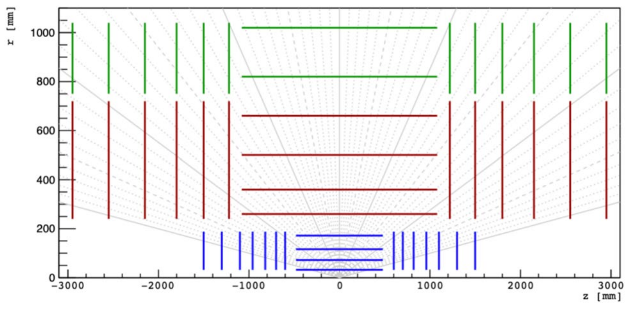


Fig. 13 Schematic layout of the TrackML detector showing the coverage of the pixel detector in blue, short strip detector in red, and long strip detector in green

Examples of a Track and Vertex Reconstruction Chain for the LHC

At the LHC, track reconstruction typically proceeds through a multi-step process, which we briefly outline here. The procedure is largely similar for different experiments, but with some key differences in strategy. For example, the CMS experiment uses an iterative tracking approach [87] in which the full track reconstruction pass is repeated a number of times, but with different configurations, and the measurements corresponding to tracks that have already been reconstructed are removed. ATLAS instead relies to a large extent on a single track reconstruction pass, but with loose track candidate search and an ambiguity resolution step to resolve between the multiple track candidates. Additional passes are used to target particular topologies, e.g., tracks produced at large radii.

As the first step, the energy deposited in the silicon detectors is grouped into *clusters* with each cluster ideally corresponding to the energy deposited by a single particle. The clusters are three-dimensional space-points formed from either a single pixel cluster or a pair of strip clusters with stereo angle between them from each side of a module, depending on the sensor technology.

Next, *seed* finding algorithms are used to reconstruct the seeds. The seeds passing a set of selection cuts are used to initiate the track finding and following algorithms, such as the CKF. After track following, the ATLAS experiment runs an ambiguity resolution algorithm to resolve duplicate tracks and remove fakes [62]. Track candidates are scored based on track properties such as the number of clusters, holes, shared clusters, and fitting quality. The scoring procedure is iterated until the selected set of track candidates are obtained. Next, the track candidates are extended into the Transition Radiation Tracker, consisting of gas-filled drift tubes, to search for additional measurements to improve the momentum resolution.

As the final track reconstruction step, if needed, a precise estimate of the track parameters is determined from *track fitting* algorithms, including the KF and Global χ^2 methods. The final track candidates are selected based on a set of track quality metrics, e.g., the number of clusters and holes, and the estimated track parameters. For example, the track candidates are usually required to satisfy a set of requirements for the momentum and impact parameters.

After track reconstruction, primary vertex candidates are reconstructed using the reconstructed tracks with estimated perigee track parameters at the beam line. Both ATLAS and CMS use an adaptive approach for primary vertex reconstruction [87, 88] similar to the ACTS Adaptive Multi-Vertex Finder (AMVF).

Track and Vertex Reconstruction Performance

As discussed in "Examples of a Track and Vertex Reconstruction Chain for the LHC", experimental applications of track reconstruction usually include many steps depending on the algorithms used, the collision environment, and the required precision. Here, an example of simplified track reconstruction chain based on a combined effort of track seeding and the CKF is discussed. The detector used for the TrackML challenge has the layout, as shown in Fig. 13, and a solenoidal magnetic field with a strength of 2 T centered on the beam line is used to demonstrate the ACTS track and vertex reconstruction performance.

Particles generated using a particle gun, both muons and charged pions, and particles from the $t\bar{t}$physics process generated in pp collisions at a center-of-mass energy of 14 TeV, the energy target for the HL-LHC, with the PYTHIA 8 generator [89, 90] are used for the performance studies. The single particle samples either contain a single particle per event for physics performance studies or a thousand particles within a single event for timing performance studies. As muons have little sensitivity to detector material, they are used to study the technical performance of the track reconstruction algorithms, while the pions are used to study the sensitivity of the track reconstruction algorithms to material. No pile-up is included in the single particle events. Two $t\bar{t}$samples are produced: one with $\langle\mu\rangle = 200$ to match the highest pile-up foreseen for the HL-LHC and the other with $\langle\mu\rangle$ varying from 0 to 300 to allow the dependence of the performance on pile-up to be studied. The interactions of the generated particles with transverse momentum, $p_T > 400$ MeV and pseudorapidity, η[5], within $|\eta| < 2.5$ with the detector are simulated with Fatras, the ACTS fast simulation library.

Detector read-out and measurement creation are detector-specific[6]; hence, a smearing algorithm applies module-specific resolutions to emulate the input measurements based on the simulated hits. The space-points constructed from the emulated measurements in the innermost four pixel layers are grouped into seeds using the ACTS seeding algorithm as described in "Track Seed Finder". Both truth and reconstructed seeds are used. Truth seeds eliminate the pattern recognition step in the seed finding, i.e., the truth information is used to identify the hits for a seed corresponding to a true particle. *Truth-generated seeds* are produced by smearing the particle properties at its point of generation. *Truth-propagated seeds* are produced by smearing the true particle information at the first detector layer. *Reconstructed seeds* are the output of the seed finding algorithm based on the simulated hits.

While each truth seed is a set of initial track parameters with associated covariance matrix, estimation of the track parameters with associated covariance matrix at the surface of the innermost space point is performed for each reconstructed seed. These initial track parameters based on either the truth seeds or the reconstructed seeds and are used to seed the CKF algorithm. After the CKF algorithm has been run, the reconstructed track candidates must satisfy a set of track quality cuts. The reconstructed tracks are required to have at least six measurements based on expectations from the TrackML detector layout and to allow initial track parameters to be located in any of the first three layers of the pixel detector. Four different types of tracks are studied, which allows the effects of the different steps in the track reconstruction sequence to be disentangled. The *truth tracks* ignore the pattern recognition entirely and are based on the properties of the simulated hits of the true particles. The *truth-generated-seeded tracks*, *truth-propagated-seeded tracks*, and *reco-seeded tracks* are reconstructed by running the CKF on the truth-generated seeds, the truth-propagated seeds, and the reconstructed seeds, respectively.

The performance of the ACTS primary vertex reconstruction module is evaluated using truth tracks with fitted perigee track parameters defined at the beam line using the same detector, magnetic field, and simulation configuration as used for the studies of the track reconstruction.

[5] Pseudorapidity is an angular quantity calculated as follows $\eta = -\ln\left[\tan\left(\frac{\theta}{2}\right)\right]$ from the polar angle θ. $\eta = \pm\infty$ corresponds to directions along the beam axis.

[6] For silicon detectors, this often requires finding connected read-out cells and either processing or emulating the detector signal.

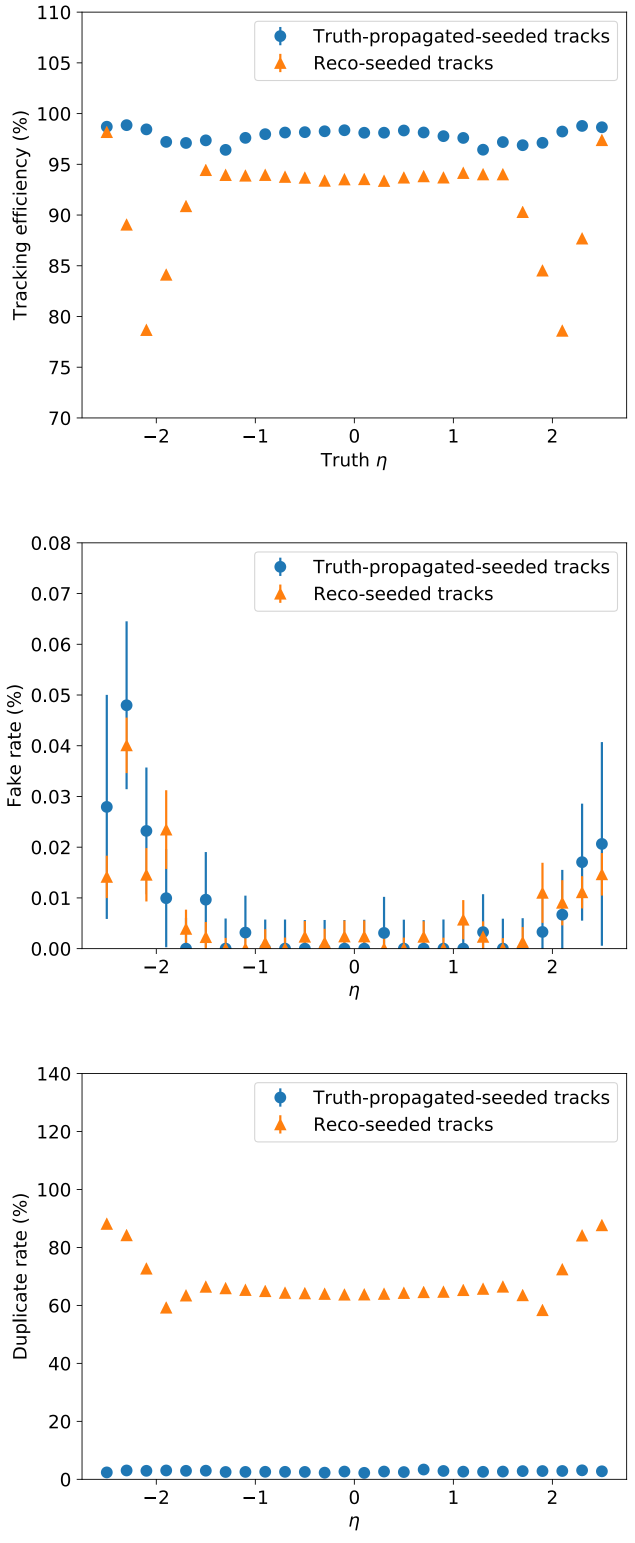


Fig. 14 The track reconstruction efficiency (top), fake rate (middle), and duplicate rate (bottom) for 1000 $t\bar{t}$ events with $\langle\mu\rangle = 200$ obtained using ACTS CKF on the TrackML detector. The blue dots and orange triangles represent results using starting parameters based on truth track parameters and those estimated from seeds found the ACTS seed finding algorithm, respectively. The truth particles used to calculate the track reconstruction efficiency are required to have $p_T > 1$ GeVand have nine measurements on the traversed detectors. The reconstructed tracks are required to have $p_T > 1$ GeVand have six measurements in the detector

Track Reconstruction Efficiency and Fake and Duplicate Rates

Key indicators of the performance of a track reconstruction algorithm are the track reconstruction efficiency, the track duplicate rate, and the rate at which fake tracks are reconstructed. Their definitions require reconstructed tracks to be associated with generated particles. A reconstructed track is associated with a generated particle if the largest fraction of measurements on the track is from this simulated particle and the fraction of associated measurements is at least 50%. A track that is not associated with any simulated particle is considered to be a fake track. Duplicate tracks occur when multiple tracks are associated with the same generated particle.

The track reconstruction efficiency is defined as the fraction of generated particles which have made at least nine measurements on the traversed detectors and are associated with tracks. The fake rate and duplicate rate of the tracks are defined as the fraction of *fake* and *duplicate* tracks among all the reconstructed tracks, respectively. Figure 14 shows the preliminary track reconstruction efficiency as a function of the η of the simulated true particle as well as the fraction of fakes and duplicates as a function of the η of the reconstructed track with the CKF for 1000 $t\bar{t}$ events with $\langle\mu\rangle = 200$. Results are shown for both the truth-propagated-seeded tracks and the reco-seeded tracks. The results for the truth-propagated-seeded tracks are excellent; however, inefficiencies and high duplicate rates are observed for the reconstructed tracks. This is because no detector-specific tuning has been performed for the TrackML detector and the performance would be improved by tuning the seed finding criteria as a function of η. The tuning of track reconstruction algorithms for a particular geometry is typically performed with several iterations and is beyond the scope of this paper.

Track Parameter Resolution

Track fitting in ACTS can be performed using either the KF or the CKF. Here, we study the track parameter resolution using the KF based on the truth tracks to remove the impact of any fake or duplicate tracks. Single muons are used to minimize the impact of detector material.

Gaussian fits are performed to the distributions of the pull values defined as the $(v_{\text{fit}} - v_{\text{truth}})/\sigma_v$. Here, v_{fit} and v_{truth} are the estimated value of the track parameter and its true simulated value, and σ_v is the estimated uncertainty of the reconstructed track parameter. The distributions of the

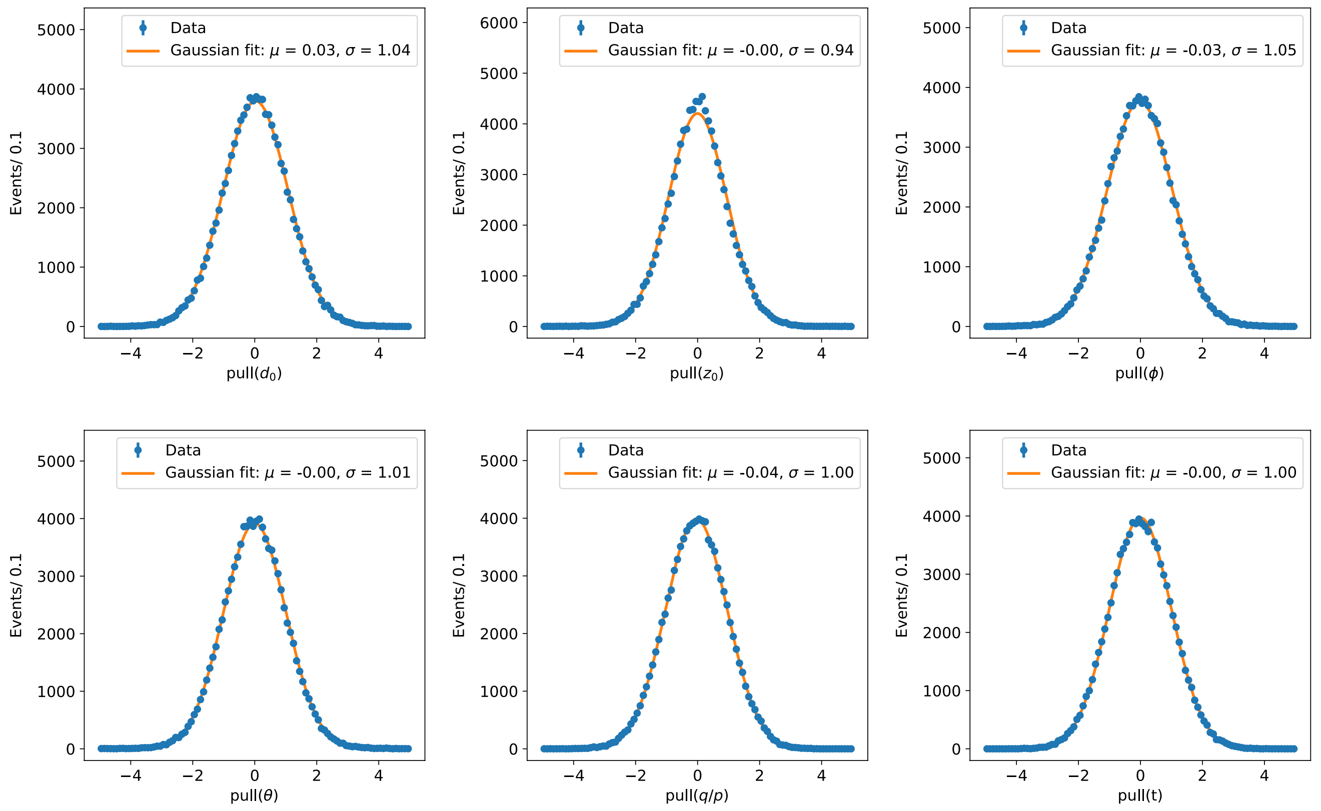


Fig. 15 The pull distributions of the six bound track parameters, d_0, z_0, ϕ, θ, $\frac{q}{p}$, and t, as obtained with the KF on the TrackML detector. The blue dots are the obtained pull values and the orange lines are the fitted Gaussian curves. For each Gaussian fit, the fitted values (with negligible uncertainties) for the parameters mean (μ) and standard deviation (σ) are shown in the legend. Truth-generated seeds are used for the KF. A sample of 100,000 single muons with 500 MeV $< p_T <$ 10 GeVand at least nine measurements on the detector is used

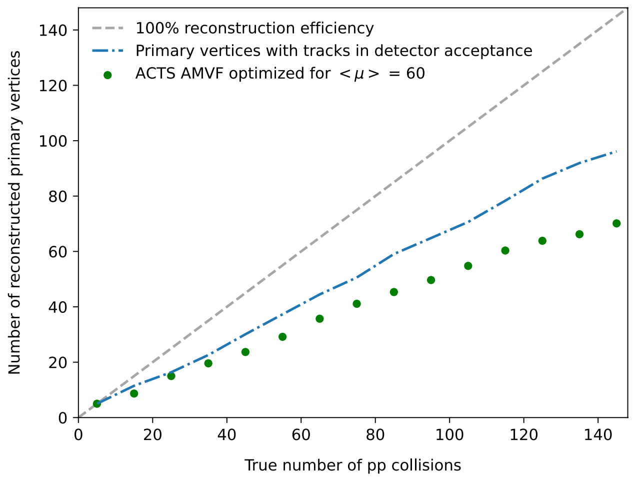


Fig. 16 Number of reconstructed primary vertices with the ACTS AMVF for different numbers of true pp collisions in simulated $t\bar{t}$ events. For reference, the gray dashed line indicates a 100% vertex reconstruction efficiency and the blue dots indicate the vertex reconstruction efficiency given a detector acceptance of $|\eta| < 2.5$ and $p_T > 400$ MeV

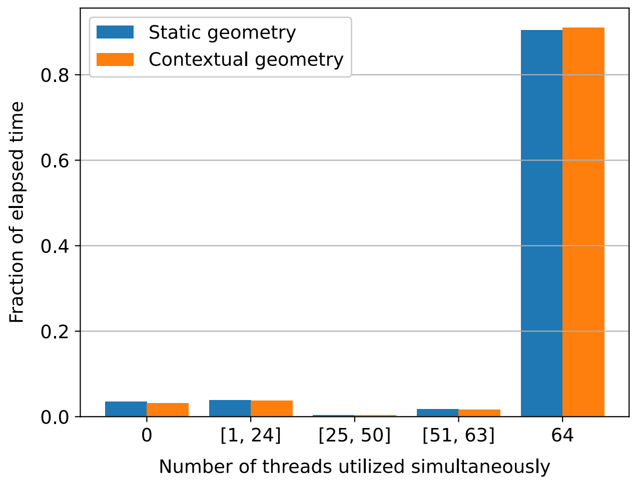


Fig. 17 The fraction of wall time during which different numbers of threads were running simultaneously while running track propagation through the TrackML detector. Either a static (blue) or contextual (orange) geometry is used for 100,000 events with 1000 pions per event using multi-threads on a Cori–Haswell node

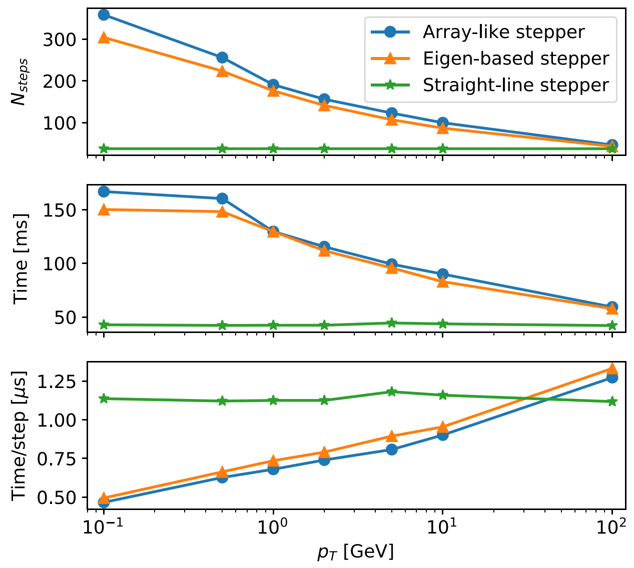


Fig. 18 The mean number of propagation steps (top), propagation time for 1000 pions (middle), and mean propagation time per step (bottom) of the track parameter propagation as a function of the p_T of the pions ($|\eta| < 2.5$) with the array-like math implementation (blue dots), the main Eigen-based stepper (orange triangles), and the straight-line stepper (green stars)

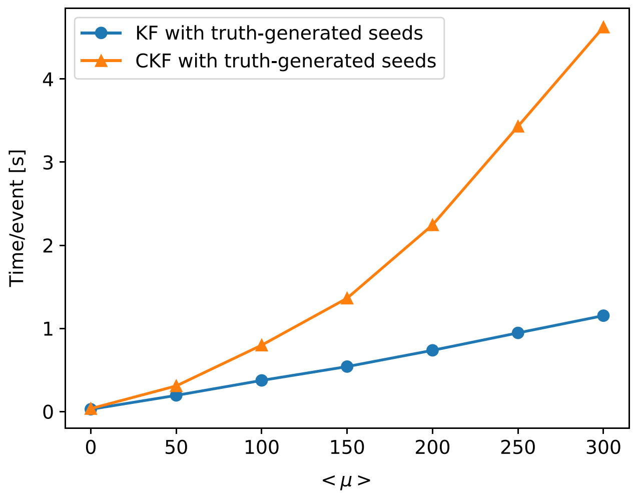


Fig. 19 The CPU time of track fitting per event using ACTS KF (blue dots) and combined track finding and track fitting (orange triangles) per event using the ACTS CKF as a function of the $\langle\mu\rangle$ of the $t\bar{t}$ sample. Truth-generated seeds are used for both the KF and the CKF. Only simulated particles with $p_T > 500$ MeVand having at least nine measurements on the detector are considered

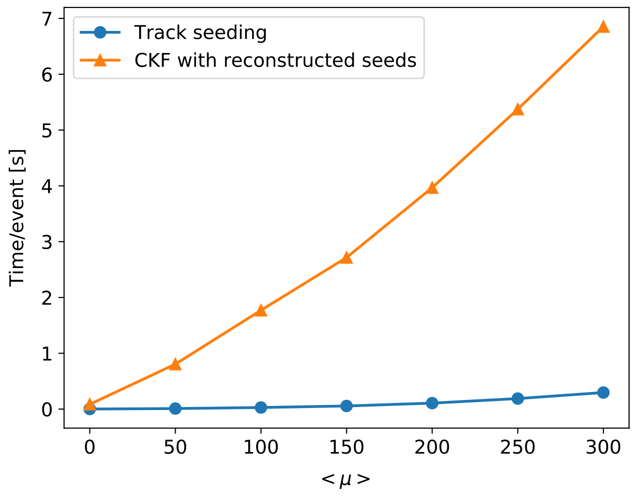


Fig. 20 The average CPU time for seed finding (blue dots) per event, and combined track finding and track fitting (orange triangles) per event with ACTS CKF using reconstructed seeds as a function of $\langle\mu\rangle$ of the $t\bar{t}$ sample. Only seeds with $p_T > 500$ MeV are considered

pulls of the track parameters at the perigee surface defined at the beam line are shown in Fig. 15. The parameters of the Gaussian distributions are approximately consistent with standard normal distributions, which demonstrates that the track parameters and their uncertainties are estimated properly by the ACTS KF. The fitted standard deviations of the Gaussian curves deviate slightly from one for the impact parameters and the momentum direction angle ϕ due to the impact of non-linear effect of the measurement model.

Primary Vertex Reconstruction Efficiency

Figure 16 shows the number of reconstructed primary vertices as a function of $\langle\mu\rangle$ of the $t\bar{t}$ sample using the ACTS AMVF based on the truth tracks. The AMVF efficiency is optimized for a mid-range working point of expected pile-up conditions for the upcoming data-taking run of the LHC, Run-3. These have $\left\langle\mu\right\rangle \approx 60$, but the performance extrapolates well to higher numbers of simultaneous *pp* interactions. When used by an experiment, the AMVF configuration would be optimized for the small pile-up range targeting the experiment's needs and accelerator conditions.

CPU Performance

The CPU performance, including the CPU utilization and time performance, was tested on a Haswell node at the National Energy Research Scientific Computing Center (NERSC) [91] (Cori-Haswell). The node has 32 physical cores and 64 threads at a clock rate of 2.3 GHz.

The TrackML detector is used to benchmark the CPU performance. The pion samples are used to evaluate the CPU utilization and the timing performance of the propagator with different numerical integration methods, and the $t\bar{t}$ samples with $\langle\mu\rangle$ varying from 0 to 300 are used to evaluate the time performance of the seed finder and CKF.

CPU Utilization

The CPU utilization of ACTS was analyzed using the `Intel VTune profiler` [92]. Figure 17 shows the CPU utilization for running track propagation through the TrackML detector with a static geometry, and a contextual geometry in which the detector alignment changes for 100,000 events with 1000 pions per event using multiple threads on the Cori–Haswell node. All 64 threads are utilized during 91% of the total execution time for both the static and the contextual geometry. This demonstrates highly efficient multi-threaded execution.

Timing Performance

Figure 18 shows the CPU time of the ACTS track parameter propagation through the TrackML detector as a function of pion p_T. Three different steppers to perform the numerical integration are shown: the main `EigenStepper` in ACTS, the stepper using manual array mathematical operations, and the straight-line stepper.

The straight-line stepper is used as the baseline, because it executes the minimal number of integration steps, even though it yields a geometrically incorrect solution, due to presence of the 2 T magnetic field. For increasing transverse momentum, the CPU time of the other two steppers approaches this theoretical best-case scenario. The other two steppers have very similar results, demonstrating that the `Eigen`-based implementation has nearly optimal computational performance.

Figure 19 shows the CPU time as a function of the $\langle\mu\rangle$ of the $t\bar{t}$ samples for track fitting with the KF, and the combined track finding and track fitting with the CKF. Both the KF and CKF are using truth-generated seeds. Only simulated particles with transverse momentum greater than 500 MeV are included. The KF and CKF require 0.74 s and 2.25 s per event, respectively, for a $t\bar{t}$ sample at $\langle\mu\rangle = 200$. The additional amount of time for the CKF is spent on the search of the compatible measurements on each measurement surface and the possible branching of the track propagation into multiple branches when more than one compatible measurements are found.

Figure 20 shows the CPU time as a function of the $\langle\mu\rangle$ of the $t\bar{t}$ samples for seed finding, and combined track finding and fitting with CKF using the reconstructed seeds. The seed finding and CKF require 0.11 s and 3.97 s per event, respectively, for a $t\bar{t}$ sample at $\langle\mu\rangle = 200$. The increase in time for the CKF in Fig. 20 compared to Fig. 19 is due to the presence of duplicate seeds among the reconstructed seeds, which would be improved by dedicated tuning of the track reconstruction algorithms.

Experience

ACTS has been initiated to preserve and evolve the well-tested track reconstruction software of the LHC era that has been used for many outstanding physics results, while also creating a research and development toolkit for algorithm optimization and design. Through the use of ACTS as the fast track simulation engine for the TrackML challenge, a wider community has become familiar with the ACTS project. The resulting R&D projects sparked by the TrackML challenge are still ongoing and have introduced new concepts and algorithms into the core ACTS software. This section discusses selected topics from experience of the development of the ACTS project.

Successful and less successful design choices typically become evident when integrating the software within experiments' software stacks. The less restrictive the initial software design, the easier such an integration is. However, this needs to be balanced against the performance of the track reconstruction software.

An example of successful design choice for ACTS is the contextual data handling: detector conditional data, such as alignment parameters, calibration constants, or other changing parameters, are usually very specific to the experiment code and a common solution for these data objects is hard to find. Due to the evolution and aging of running experiments, details of calibration data may not necessarily be known when an experiment begins. Within ACTS, the implementation of the contextual data and the data flow through the software are split. This allows experiments to implement specific data objects for conditions and the ACTS software handles them throughout the entire call chain. This also allows the conditions to be unpacked at the appropriate time by the detector-specific code. As the object type is known and specified on both ends of the call chain, this guarantees minimum conversion overhead. The contextual data handling was first demonstrated within the ACTS examples, and has also been demonstrated while integrating ACTS in Athena.

A similar example of a successful design choice is the implementation of screen logging. Messages output on the screen are commonly for debugging and quality control with particle and nuclear physics software; hence, a seamless integration of ACTS with the experiments logging infrastructure has been a priority during development. The integration has been achieved by allowing the logging instance in ACTS to be replaced with a custom logger connected to the experiments framework logging facility. The logging has been proven to work within the Gaudi-based software frameworks of ATLAS and FCC-hh. In addition, a generic demonstrator showing how to change the logging instance is included in the ACTS test suite.

The choice of using C++ was easy, given the current landscape of particle and nuclear physics software. C++ is an extremely powerful language, but comes, like any language choice, with its shortcomings. Initially, extensive use of template expressions in the ACTS core software led to huge resource requirements during compilation. Therefore, this has been revised to reduce the resource requirements. Care is needed to maintain simplicity within the code, which will also be important for an eventual re-use of parts of the ACTS software on heterogeneous hardware. While writing code for heterogeneous hardware has not been an immediate target of the ACTS project, compatibility should be foreseen, allowing ACTS to adapt to future particle and nuclear physics computing landscapes.

Given that the origin of ACTS lies in the ATLAS Common Tracking Software, several initial design choices focused towards general-purpose collider experiments. Weaknesses relating to the use of ACTS for different geometry types, particularly for forward detectors, time projection chambers, drift tube, and telescope setups, have been identified. While some of them have already been resolved, these remain active areas of development.

Conclusion

The development of efficient and maintainable track reconstruction is a challenge for current and future particle and nuclear physics experiments. We have introduced the ACTS toolkit, which provides a set of open-source, experiment-independent and framework-independent reconstruction algorithms for tracking detectors. The high-level track reconstruction tools do not depend on the details of the detection technologies and magnetic field configuration, and have been tested for thread-safety to support concurrent event processing.

We have demonstrated that ACTS is maturing as a toolkit and currently provides a range of algorithms for track and vertex reconstruction, which have been or are actively being integrated into a range of experimental frameworks. Geometries for a number of particle and nuclear physics detectors have been included in ACTS. Initial studies of the physics and computational performance of the track and vertex reconstruction algorithms using the TrackML detector were presented. A detailed tuning of the algorithms is required to achieve full performance for any specific detector, which is beyond the scope of this paper and is expected to be performed by the individual experimental collaborations.

A discussion of selected experiences obtained during the ACTS project has been presented. Future development directions for the ACTS project are expected to include further deployment into experimental frameworks and increasing investment into the R&D lines. The authors would welcome contact from other experiments interested in exploring the use of and contributions to ACTS.

Acknowledgements This work was supported by the CERN Strategic R&D Program on Technologies for Future Experiments (CERN-OPEN-2018-006), the National Science Foundation under Cooperative Agreement OAC-1836650, the Office of Nuclear Physics within the U.S. DOE Office of Science under Contract No. DE-SC0012704, and the German Federal Ministry of Education and Research (BMBF).

Funding Open access funding provided by CERN (European Organization for Nuclear Research). This work was supported by the CERN Strategic R&D Program on Technologies for Future Experiments (CERN-OPEN-2018-006), the National Science Foundation under Cooperative Agreement OAC-1836650, the Office of Nuclear Physics within the U.S. DOE Office of Science under Contract No. DE-SC0012704, and the German Federal Ministry of Education and Research (BMBF).

Data Availability Statement This manuscript has no associated data or the data will not be deposited. [Authors' comment: There are no associated data available.]

Code Availability The code used for this research is available open source [63].

Declarations

Conflict of Interest The authors declare that they have no conflict of interest.

References

1. ATLAS Collaboration. The ATLAS experiment at the CERN large hadron collider. In: JINST 3 (2008), p S08003. https://doi.org/10.1088/1748-0221/3/08/S08003
2. Apollinari G et al (2017) High-luminosity large hadron collider (HL-LHC): technical design report V. 0.1. https://doi.org/10.23731/CYRM-2017-004
3. CMS Collaboration (2008) The CMS experiment at the CERN LHC. In: JINST 3, p S08004. https://doi.org/10.1088/1748-0221/3/08/s08004
4. CMS Collaboration (2021) Evolution of the CMS computing model towards phase-2. Technical report. Geneva: CERN. https://cds.cern.ch/ record/2751565
5. Calafiura P et al (2020) ATLAS HL-LHC Computing conceptual design report. Technical report CERN-LHCC-2020-015, LHCC-G-178. CERN, Geneva. http://cds.cern.ch/record/2729668

6. Collaboration FCC (2019) FCC-hh: the hadron collider. Eur Phys J ST 228(4):755–1107. https://doi.org/10.1140/epjst/e2019-900087-0
7. HEP-SPEC06 Benchmark. https://w3.hepix.org/benchmarking.html. Accessed 17 May 2021
8. Moore GE (1965) Cramming more components onto integrated circuits. Electronics 38:8. https://doi.org/10.1109/JPROC.1998.658762
9. Hennessy JL, Patterson DA (2019) A new golden age for computer architecture. Commun ACM 62(2):48–60. https://doi.org/10.1145/3282307
10. Pantaleo F (2017) New track seeding techniques for the CMS experiment. PhD thesis. Hamburg U. https://cds.cern.ch/record/2293435
11. Ju X et al (2020) Graph neural networks for particle reconstruction in high energy physics detectors. In: NeurIPS 2019. arXiv: 2003.11603 [physics.ins-det]
12. Amrouche S et al (2019) Similarity hashing for charged particle tracking. In: IEEE international conference on big data 2019, pp 1595–1600. https://doi.org/10.1109/BigData47090.2019.9006316
13. Strandlie A, Frühwirth R (2010) Track and vertex reconstruction: from classical to adaptive methods. Rev Mod Phys 82:1419–1458. https://doi.org/10.1103/RevModPhys.82.1419
14. Hough P (1959) Machine analysis of bubble chamber pictures. In: Kowarski L (eds) Conference proceedings C 590914, pp 554–558
15. Duda R, Hart P (1972) Use of the Hough transformation to detect lines and curves in pictures. Commun ACM 15:11–15
16. Hopfield J (1982) Neural networks and physical systems with emergent collective computational abilities. Proc Natl Acad Sci 79:2554–2558. https://doi.org/10.1073/pnas.79.8.2554
17. Kalman RE (1960) A new approach to linear filtering and prediction problems. J Basic Eng 82(1):35–45. https://doi.org/10.1115/1.3662552
18. Billoir P (1984) Track fitting with multiple scattering: a new method. Nucl Instrum Methods A 225:352–366. https://doi.org/10.1016/0167-5087(84)90274-6
19. Frühwirth R (1987) Application of Kalman filtering to track and vertex fitting. Nucl Instrum Methods A 262:444–450. https://doi.org/10.1016/0168-9002(87)90887-4
20. Gumpert C et al (2017) ACTS: from ATLAS software towards a common track reconstruction software. Technical report. ATL-SOFT-PROC-2017-030. 4. CERN, Geneva. https://doi.org/10.1088/1742-6596/898/4/042011
21. Ai X (2019) Acts: a common tracking software'. In: Meeting of the division of particles and fields of the American Physical Society. arXiv: 1910.03128 [physics.ins-det]
22. Ai X (2020) Tracking with a common tracking software. arXiv: 2007.01239 [physics.ins-det]
23. Gessinger P et al (2020) The Acts project: track reconstruction software for HL-LHC and beyond. In: Doglioni C (eds) EPJ web conference 245: 10003. https://doi.org/10.1051/epjconf/202024510003
24. Amrouche S et al. (2020) The tracking machine learning challenge: accuracy phase. In: The NeurIPS '18 competition, pp 31-264. https://doi.org/10.1007/978-3-030-29135-8_9
25. Kiehn M et al (2019) The TrackML high-energy physics tracking challenge on Kaggle. In: Forti A (eds) EPJ web conference 214: 06037. https://doi.org/10.1051/epjconf/201921406037
26. Amrouche S et al (2021) The tracking machine learning challenge: throughput phase. arXiv: 2105.01160 [cs.LG]
27. Tüysüz C et al (2020) Performance of particle tracking using a quantum graph neural network. In: BASARIM 2020 conference proceedings. arXiv: 2012.01379 [quant-ph]
28. Fox PJ et al (2020) Beyond 4D tracking: using cluster shapes for track seeding. arXiv: 2012.04533 [physics.ins-det]
29. Heintz A et al (2020) Accelerated charged particle tracking with graph neural networks on FPGAs. In: NeurIPS 2020. arXiv: 2012.01563 [physics.ins-det]
30. Bapst F et al (2020) A pattern recognition algorithm for quantum annealers. In: Comput Softw Big Sci 4(1) 1. https://doi.org/10.1007/s41781-019-0032-5. arXiv: 1902.08324 [quant-ph]
31. Abe T et al (2010) Belle II technical design report. arXiv: 1011.0352 [physics.ins-det]
32. The CEPC Study Group (2018) CEPC conceptual design report: volume 1—accelerator. arXiv: 1809.00285 [physics.acc-ph]
33. The CEPC Study Group (2018) CEPC conceptual design report: volume 2—physics and detector. arXiv: 1811.10545 [hep-ex]
34. Adare A et al (2015) An upgrade proposal from the PHENIX collaboration. arXiv: 1501.06197 [nucl-ex]
35. sPHENIX Collaboration (2020) Requirements, status, and plans for track reconstruction at the sPHENIX experiment. In: Proceedings for the connecting the dots workshop. arXiv: 2007.00771 [physics.ins-det]. Accessed 24 June 2021
36. sPHENIX Collaboration (2021) Implementation of ACTS into sPHENIX track reconstruction. In: 25th International conference on computing in high-energy and nuclear physics. arXiv: 2103.06703 [physics.ins-det]
37. PANDA Collaboration (2012) Technical design report for the: PANDA micro vertex detector. Technical report. arXiv:1207.6581 [physics.ins-det]
38. Bettoni D (2007) The PANDA experiment at FAIR. In: Mahlke H, Napolitano J (eds) eConf C070805, p 39. arXiv: 0710.5664 [hep-ex]
39. FASER Collaboration (2019) FASER: ForwArd Search ExpeRiment at the LHC. arXiv: 1901.04468 [hep-ex]
40. ATLAS Collaboration (2017) Technical design report for the ATLAS inner tracker pixel detector. Technical report CERN-LHCC-2017- 021, ATLAS-TDR-030. https://cds.cern.ch/record/2285585
41. ATLAS Collaboration (2017) Technical design report for the ATLAS inner tracker strip detector. Technical report CERN-LHCC-2017-005, ATLAS-TDR-025. https://cds.cern.ch/record/2257755
42. ATLAS Collaboration (2019) Expected tracking performance of the ATLAS inner tracker at the HL-LHC. Technical report. Geneva: CERN. https://cds.cern.ch/record/2669540
43. Ai X et al (2020) Acts project: v3.0.0. version v3.0.0. Nov. https://doi.org/10.5281/zenodo.3937454. Accessed 24 June 2021
44. Barrand G et al (2001) GAUDI–a software architecture and framework for building HEP data processing applications. Comput Phys Commun 40:45–55. https://doi.org/10.1016/S0010-4655(01)00254-5
45. LHCb Collaboration (2008) The LHCb detector at the LHC. In: JINST 3, S08005. https://doi.org/10.1088/1748-0221/3/08/s08005
46. Athena (2019) Version 22.0.1. https://doi.org/10.5281/zenodo.2641996. Accessed 24 June 2021
47. CMS Offine Software. http://cms-sw.github.io/. Accessed 02 Mar 2021
48. Brun R, Gheata A, Gheata M (2003) The ROOT geometry package. Nucl Instrum Methods Phys Res A 502(2):676–680. https://doi.org/10.1016/S0168-9002(03)00541-2
49. Threading building blocks. https://github.com/oneapi-src/oneTBB. Accessed 03 Feb 2021
50. Ai X et al (2021) A GPU-based Kalman Filter for track fitting. arXiv: 2105. 01796 [physics.ins-det]
51. Amrouche S et al (2021) Hashing and metric learning for charged particle tracking. In: 33rd Annual conference on neural information processing systems. arXiv: 2101.06428 [hep-ex]
52. Open neural network exchange. https://www.github.com/onnx. Accessed 17 May 2021
53. autodiff. https://autodiff.github.io/. Accessed 02 Mar 2021

54. Nickolls J et al (2008) Scalable parallel programming with CUDA: is CUDA the parallel programming model that application developers have been waiting for? Queue 6(2):40–53. https://doi.org/10.1145/1365490.1365500
55. SYCL: C++ single-source heterogeneous pro-gramming for OpenCL. https://www.khronos.org/sycl/. Accessed 02 Mar 2021
56. ATLAS Collaboration (2005) ATLAS computing: technical design report. Technical report. Geneva: CERN. http://cds.cern.ch/record/ 837738
57. CMS Collaboration (2005) CMS: the computing project. Technical design report. Technical report. Geneva: CERN. http://cds.cern.ch/record/838359
58. ISO/IEC 14882:2017 (2017) Programming languages - C++. Standard. International Organization for Standardization, Geneva. https://www.iso.org/standard/68564.html. Accessed 17 May 2021
59. Guennebaud G, Jacob B et al (2010) Eigen v3. http://eigen.tuxfamily.org
60. Boost. https://www.boost.org. Accessed 02 Mar 2021
61. CMake. https://cmake.org. Accessed 02 Mar 2021
62. Cornelissen T et al (2008) The new ATLAS track reconstruction (NEWT). J Phys Conf Ser 119:3. https://doi.org/10.1088/1742-6596/119/3/032014
63. ACTS on Github. https://github.com/acts- project/acts. Accessed 05 Apr 2021
64. Edmonds K et al (2008) The fast ATLAS track simulation (FATRAS). Technical report ATL-SOFTPUB- 2008-001. ATL-COM-SOFT-2008-002. Geneva: CERN. https://cds.cern.ch/record/1091969
65. Brun R, Rademakers F (1997) ROOT: an object oriented data analysis framework. In: Werlen M, Perret-Gallix D (eds) Nuclear instruments and Methods A 389, pp 81-86. https://doi.org/10.1016/S0168-9002(97)00048-X
66. Semantic versioning. https://semver.org/. Accessed 17 May 2021
67. Mozilla Public License Version 2. https://www.mozilla.org/en-US/MPL/2.0/. Accessed 02 Mar 2021
68. clang-format. https://clang.llvm.org/docs/ClangFormat.html. Accessed 14 June 2021
69. The LLVM compiler infrastructure. https://www.llvm.org. Accessed 14 June 2021
70. GEANT4 Collaboration (2003) GEANT4: a simulation toolkit. Nucl Instrum Methods A 506:250–303. https://doi.org/10.1016/S0168-9002(03)01368-8
71. Petrič M et al (2017) Detector simulations with DD4hep. J Phys Conf Ser. https://doi.org/10.1088/1742-6596/898/4/042015
72. Tsulaia V, Boudreau J (2004) The GeoModel Toolkit for detector description. https://indico.cern.ch/event/0/contributions/1294152/. CHEP 2004 (Interlaken)
73. Myrheim J, Bugge L (1979) A fast Runge-Kutta method for fitting tracks in a magnetic field. Nucl Instrum Methods 160(1):43–48. https://doi.org/10.1016/0029-554X(79)90163-0
74. Lund E et al (2009) Track parameter propagation through the application of a new adaptive Runge-Kutta-Nyström method in the ATLAS experiment. In: JINST 4: P04001. https://doi.org/10.1088/1748-0221/4/04/P04001
75. Rauch HE, Tung F, Striebel CT (1965) Maximum likelihood estimates of linear dynamic systems. AIAA J 3(8):1445–1450. https://doi.org/10.2514/3.3166
76. Billoir P (1989) Progressive track recognition with a Kalman-like fitting procedure. Comput Phys Commun 57(1):390–394. https://doi.org/10.1016/0010-4655(89)90249-X
77. Billoir P, Qian S (1990) Simultaneous pattern recognition and track fitting by the Kalman filtering method. Nucl Instrum Methods Phys Res A 294(1):219–228. https://doi.org/10.1016/0168-9002(90)91835-Y
78. Mankel R (1997) A concurrent track evolution algorithm for pattern recognition in the HERA-B main tracking system. Nucl Instrum Methods Phys Res A 395(2):169–184. https://doi.org/10.1016/S0168-9002(97)00705-5
79. ATLAS Collaboration (2019) Development of ATLAS primary vertex reconstruction for LHC Run 3. Technical report. ATL-PHYS-PUB-2019-015. CERN, Geneva. https://cds.cern.ch/record/2670380
80. Bickel DR, Frühwirth R (2006) On a fast, robust estimator of the mode: comparisons to other robust estimators with applications. Comput Stat Data Anal 50(12):3500–3530. https://doi.org/10.1016/j.csda.2005.07.011
81. Billoir P, Qian S (1992) Fast vertex fitting with a local parametrization of tracks. Nucl Instrum Methods Phys Res A 311(1):139–150. https://doi.org/10.1016/0168-9002(92)90859-3
82. Piacquadio G (2010) Identification of b-jets and investigation of the discovery potential of a Higgs boson in the $WH \rightarrow l\nu b\bar{b}$ channel with the ATLAS experiment. PhD thesis. Freiburg U. https://cds.cern.ch/record/1243771
83. ATLAS Collaboration (2020) Technical design report: a high-granularity timing detector for the ATLAS phase-II upgrade. Technical report. CERN-LHCC-2020-007; ATLAS-TDR-031. Geneva: CERN. https://cds.cern.ch/record/2719855. Accessed 24 June 2021
84. SuperKEKB Collaboration (2018) SuperKEKB collider. Nucl Instrum Methods A 907: 188-199. https://doi.org/10.1016/j.nima.2018.08.017. arXiv: 1809.01958 [physics.acc-ph]
85. Belle II Tracking Group Collaboration (2021) Track finding at Belle II. Comput Phys Commun 259:107610. https://doi.org/10.1016/j.cpc.2020.107610. arXiv: 2003.12466 [physics.ins-det]
86. Allaire C et al (2021) OpenDataDetector. Version v1. https://doi.org/10.5281/zenodo.4674401. Accessed 24 June 2021
87. CMS Collaboration (2014) Description and performance of track and primary-vertex reconstruction with the CMS tracker. JINST 9(10):P10009. https://doi.org/10.1088/1748-0221/9/10/P10009
88. Borissov G et al (2015) ATLAS strategy for primary vertex reconstruction during Run-2 of the LHC. J Phys Conf Ser. https://doi.org/10.1088/1742-6596/664/7/072041
89. Sjöstrand T, Mrenna S, Skands P (2008) A brief introduction to PYTHIA 8.1. Comput Phys Commun 178(11):852–867. https://doi.org/10.1016/j.cpc.2008.01.036
90. Sjöstrand T, Mrenna S, Skands P (2006) PYTHIA 6.4 physics and manual. J High Energy Phys 05:026. https://doi.org/10.1088/1126-6708/2006/05/026
91. NERSC Cori system specification. https://docs.nersc.gov/systems/cori/#system-specification. Accessed 02 Apr 2021
92. Intel VTune Profiler. https://software.intel.com/content/www/us/en/develop/tools/vtune-profiler.html. Accessed 17 May 2021