Development and improvement of track reconstruction software and search for disappearing tracks with the ATLAS experiment

Dissertation
zur Erlangung des Grades

Doktor der Naturwissenschaft

am Fachbereich Physik, Mathematik und Informatik
der Johannes Gutenberg-Universität
in Mainz

von
Paul Gessinger-Befurt
geboren in Mainz

Mainz, den 10. Mai 2021

Datum der mündlichen Prüfung: 30. April 2021

Kurzfassung

Hochenergie-Teilchenphysik beschäftigt sich mit der fundamentalsten Art, auf welche die Gesetze der Natur verstanden werden können. Mit künstlich erzeugten Teilchenkollisionen ist es möglich, die Eigenschaften von Teilchen und deren Wechselwirkungen zu vermessen. Das Standardmodell der Teilchenphysik ist eine Kombination von Theorien, welche drei der vier fundamentalen Wechselwirkungen beschreiben können. Es dient als Grundlage zur Interpretation von Beobachtungen in der Teilchenphysik. Das Standardmodell erlaubt sehr präzise Vorhersagen und Berechnungen. Dennoch gilt es als unvollständige Theorie, da manche Phänomene, wie die Existenz von Dunkler Materie oder Neutrinooszillationen, unerklärt bleiben.

Die Analyse von Teilchenkollisionen geschieht durch Messung der Teilchen, die in den Kollisionen produziert werden. Eine wichtige Komponente hierfür ist die Rekonstruktion von Spuren geladener Teilchen, sogenannter *Tracks*. Spezielle Sensoren sind in der Lage, Messungen entlang der Teilchenspuren vorzunehmen. Durch Kombination dieser Messungen ist es möglich, die Spuren mithilfe von Computerprogrammen zu rekonstruieren. Diese Rekonstruktion stellt große Herausforderungen an die eingesetzten Algorithmen dar, aufgrund ihrer Abhängigkeit von der Komplexität der Ereignisse. Zukünfige Erhöhungen der instantanen Luminosität erfordern daher weitere Verbesserungen dieser Algorithmen.

Ein Teil dieser Arbeit beschäftigt sich mit der Entwicklung und Verbesserung der Software, die für die Spurrekonstruktion verwendet wird. Sie leistet somit einen Beitrag, die oben genannten Herausforderungen anzugehen. Dies umfasst sowohl die Entwicklung der Software des ATLAS Experiments, als auch die Arbeit an einer vom Experiment unabhängigen Software namens ACTS. Einzelne Beiträge in beiden Bereichen werden im Detail diskutiert und erklärt. Ein Beispiel hierfür ist die Reintegration von ACTS Komponenten in ATLAS, insbesondere die Modellierung der Geometrie des ATLAS Spursystems. Des Weiteren findet sich die Beschreibung eines Ansatzes zur Behandlung von kleinen Verschiebungen der einzelnen Sensoren im Kontext einer parallelisierten Programmausführung. Außerdem wird eine Neuentwicklung der Datenstruktur, die für die Filterung von Teilchenspuren in ACTS verwendet wird, besprochen.

Ein weiterer Teil dieser Arbeit beschreibt eine konkrete Anwendung von Teilchenspurrekonstruktion. Eine Analyse von ATLAS Daten aus Proton-Proton Kollisionen bei $\sqrt{s} = 13\,\text{TeV}$ in Form einer Suche nach einer Erweiterung des Standardmodells wurde durchgeführt. Diese Erweiterung sagt die Existenz von langlebigen Teilchen vorher, die auf dem Weg durch das ATLAS Spursystem zerfallen. Da diese geladenen Teilchen vor ihrem Zefall bereits Spursignale hinterlassen haben, erzeugen sie sogenannte *verschwindende Spuren*. Die hier beschriebene Analyse macht sich kürzlich verfügbar gewordene Verbesserungen in den dedizierten Algorithmen für die Rekonstruktion dieser besonders kurzen Spuren zunutze. Sie beinhaltet die erwartete Sensitivität für dieses Signalmodell mit den existierenden Daten, welche in einer erhöhten erwarteten Massenausschlussgrenze resultiert. Zusätzlich findet sich eine Vorhersage der Sensitivität mit einem größeren Datensatz.

Abstract

High-energy particle physics concerns itself with the most fundamental level at which the laws of nature can be understood. Using particle collisions, it is possible to probe and measure the properties of particles and their interactions. The Standard Model of particle physics is a set of theories describing three of the four fundamental interactions, and is the baseline with which observations are interpreted. Even though the Standard Model allows precise calculations of particle phenomena, it is thought to be incomplete, as certain observations like dark matter or neutrino oscillations remain unexplained.

The analysis of particle collisions requires the measurement of particles produced in these collisions. One major part of these measurements is the reconstruction of the trajectories of charged particles, called *tracks*. Dedicated sensitive elements are used to obtain measurements of the particle along its trajectory, in order to ultimately reconstruct the track using software algorithms. Track reconstruction is a complex application, which grows in complexity with event activity. Therefore, future increases of the instantaneous luminosity of the LHC pose a challenge and require advancements in both the computational and physics peformance of track reconstruction algorithms.

One part of this thesis presents work toward the development and improvement of track reconstruction in the context of the ATLAS experiment, and an experiment-independent software toolkit called ACTS. Additionally, the effort to use ACTS components in the ATLAS software is described. Specific contributions that were made to both domains are shown, which address the aforementioned challenge. Prominent examples are the description of current and future ATLAS tracker geometries, concurrent handling of misalignments, and the improvement of data structures used for reconstruction.

Another part of this thesis describes a concrete application of these reconstructed particle tracks, in the form of an analysis of data from proton-proton collisions at $\sqrt{s} = 13\,\text{TeV}$ recorded by ATLAS, searching for an extension of the Standard Model. In this theoretical scenario, particles travel through the innermost part of the tracker, the Pixel detector, before decaying, resulting in so-called *disappearing tracks*. The analysis described here uses recent developments in the dedicated techniques used to reconstruct these short tracks to evaluate their expected sensitivity. An increase in expected sensitivity in the form of a higher expected mass limit is found to result from inclusion of new shorter tracks than were used in previous analyses. Potential for further developments is discussed in light of the LHC upgrade, and future colliders.

Contents

Kurzfassung . . . iii
Abstract . . . v

1. Introduction **1**

I. Background of theory and experimental techniques **5**

2. Theoretical background **7**
2.1. The Standard Model of Particle Physics . . . 7
2.1.1. Particle content . . . 8
2.1.2. Mathematical structure . . . 9
2.1.3. Quantities of interest in collider events . . . 14
2.1.4. Proton-proton interactions . . . 16
2.2. Outstanding issues in the Standard Model . . . 20
2.3. Supersymmetry . . . 21
2.3.1. MSSM and compressed particle mass spectra . . . 23

3. The ATLAS experiment **25**
3.1. The Large Hadron Collider . . . 25
3.2. Experimental setup and detector components . . . 27
3.2.1. Coordinate system . . . 27
3.2.2. Tracking system . . . 29
3.2.3. Calorimeters . . . 32
3.2.4. Muon system . . . 35
3.2.5. Solenoid and toroid magnets . . . 36
3.3. Computing . . . 37
3.3.1. Online computing . . . 37
3.3.2. Offline computing . . . 37
3.4. Trigger and data acquisition . . . 39
3.4.1. Trigger system . . . 39
3.4.2. Missing transverse energy triggers . . . 41
3.4.3. Data acquisition system . . . 42
3.5. Event reconstruction . . . 43
3.5.1. Reconstruction of detector signatures . . . 43
3.5.2. Reconstruction of physics signatures . . . 44

3.5.3. Particle identification and isolation 46
3.5.4. Missing transverse momentum 47
3.6. Simulation 48
3.6.1. Simulation of processes 48
3.6.2. Simulation of detector response 49
3.7. Calibration and corrections 49
3.7.1. Calibration 49
3.7.2. Efficiency corrections 50
3.7.3. Pile-up effects 50
3.8. Luminosity measurement 51

4. Reconstruction of charged particle trajectories 53
4.1. Charged particle detection 53
4.2. Track parametrization 55
4.3. Particle propagation 57
4.3.1. Numerical integration 57
4.3.2. Covariance transport 59
4.3.3. Material effects 59
4.4. Geometry and material modelling 60
4.5. Clusterization 64
4.6. Space point formation and seeding 66
4.7. Track finding and track fitting 68
4.7.1. The Kalman formalism 69
4.7.2. Combinatorial Kalman Filter 72
4.8. Ambiguity resolution 72
4.9. Vertex reconstruction 74
4.10. Alignment of particle sensors 75
4.11. Summary 77

5. HL-LHC and challenges of the High-Luminosity era 79
5.1. The High-Luminosity-LHC 79
5.2. Software and computing challenges 80
5.3. Tracking in ATLAS at the HL-LHC 82
5.3.1. ATLAS Phase-2 Upgrade Inner Tracker 82
5.3.2. Track reconstruction with the ITk 84

II. Development and improvement of track reconstruction software 89

6. The ACTS project 91
6.1. Overview 91
6.1.1. Origins and objectives 91
6.1.2. Strategy and goals 92

6.1.3. Code maintainability . . . 93
6.2. Thread-safety . . . 96
6.3. Components . . . 98
6.3.1. Geometry description . . . 100
6.3.2. Event Data Model . . . 101
6.3.3. Particle propagation . . . 103
6.3.4. Magnetic field lookup . . . 104
6.4. Software development best practices and implementation . . . 107
6.4.1. Build system and version control . . . 107
6.4.2. Unit and integration testing . . . 108
6.4.3. Continuous integration . . . 109
6.5. Recent developments . . . 109

7. Development and improvement of the ACTS software 111
7.1. Integration of ACTS into the ATLAS software framework . . . 111
7.1.1. ATLAS environment . . . 111
7.1.2. ACTS specific packages . . . 112
7.2. Description of current ATLAS geometry . . . 114
7.2.1. GeoModel geometry . . . 114
7.2.2. Readout geometry . . . 116
7.2.3. Tracking geometry . . . 117
7.2.4. Extrapolation test . . . 122
7.3. Construction of ACTS tracking geometry for ITk . . . 124
7.3.1. Review of description of endcap SCT modules . . . 127
7.3.2. Review of cartesian coordinate system for ITk endcap sensors . . . 130
7.3.3. Polar coordinate system for ITk endcap sensors . . . 133
7.3.4. Summary . . . 139
7.4. Implementation of a fast navigation system through arbitrary geometries . . . 140
7.4.1. Conventional navigation model . . . 141
7.4.2. Bounding boxes . . . 145
7.4.3. Ray and Frustum intersections with boxes . . . 146
7.4.4. Bounding Volume Hierarchy navigation model . . . 151
7.4.5. Performance characterization . . . 155
7.4.6. Summary . . . 163
7.5. Design and improvement of the ACTS event data model . . . 164
7.5.1. Compile time calculations and optimization . . . 165
7.5.2. Column-based storage model for filtering applications . . . 172
7.6. Memory management of shared objects . . . 182
7.7. Concurrent handling of time dependent parameters such as alignment . . . 185
7.7.1. Time dependent misalignment . . . 185
7.7.2. Conditions handling in ATLAS . . . 186

7.7.3. Concurrent handling of alignment corrections . . . 187
7.7.4. Component structure . . . 194
7.8. Summary and outlook on work to be done . . . 195

III. Search for long-lived particles presenting as disappearing tracks 199

8. Disappearing tracks in the ATLAS detector 201
8.1. The disappearing track signature . . . 201
8.2. Four- and three-layer tracklet reconstruction . . . 202
8.2.1. Vertex constrained tracklet fit . . . 206
8.2.2. Tracklet properties . . . 210

9. Search for long-lived charginos in the ATLAS detector 217
9.1. Overview and analysis strategy . . . 217
9.2. Signal model . . . 218
9.2.1. Aspects and characteristics . . . 218
9.2.2. Existing results of searches . . . 221
9.3. Background sources . . . 222
9.3.1. Charged lepton scatter background . . . 222
9.3.2. Hadron scatter background . . . 224
9.3.3. Combinatorial fake background . . . 224
9.4. Datasets and simulated samples . . . 225
9.4.1. Recorded dataset . . . 225
9.4.2. Simulated samples . . . 228
9.5. Object definition . . . 229
9.5.1. Tracklets . . . 229
9.5.2. Electrons . . . 232
9.5.3. Muons . . . 232
9.5.4. Jets . . . 233
9.5.5. Overlap removal . . . 233
9.5.6. Missing transverse energy . . . 234
9.6. Signal event selection . . . 234
9.7. Template based background estimation . . . 237
9.7.1. Lepton and hadron control regions . . . 237
9.7.2. Scattered lepton template . . . 241
9.7.3. Scattered hadron template . . . 247
9.7.4. Combinatorial fake template . . . 247
9.7.5. Background normalization . . . 248
9.8. Unified total background estimation . . . 250
9.9. Systematic uncertainties . . . 252

9.10. Results and statistical interpretation 255
9.10.1. Likelihood-based hypothesis tests 256
9.10.2. Expected signal sensitivity 260
9.11. Summary and outlook toward further improvements 268
9.11.1. Improved estimation of backgrounds 269
9.11.2. Reconstruction of soft secondary particles and vertex tagging 270
9.11.3. Future detectors and colliders 272

10. Conclusion **273**

A. ACTS development **279**

B. Disappearing track search **283**
B.1. Systematic uncertainties 286
B.2. Pre-fit and post-fit distributions 293
B.2.1. Individual fits 293
B.2.2. Combined fit 301
B.3. Nuisance parameter pulls 309
B.4. Signal strength upper limits 309

Bibliography **313**

1. Introduction

Particle physics at highest energies is a means to probe nature and its phenomena at the most fundamental level. The elementary building blocks of matter and the interactions between them are situated at the very center of an understanding of the world. No picture of the universe is complete without a thorough grasp of the properties at the microscopic level.

The history of high-energy particle physics has seen constant efforts to push the energy frontier to new levels. With the introduction of particle colliders, observation of particle collisions became an incredibly useful tool to measure a large variety of aspects of the particle domain. Over the course of the 20th century, a progressively more comprehensive picture was formed. Using methodologies of Quantum Field Theory (QFT), electromagnetic particle interactions could be described in a coherent fashion. After the discovery of a series of particles that were believed to be elementary, these turned out to be composite in nature. Alongside this realization, a quantized description of the *strong* interaction binding them together emerged, and was subsequently confirmed experimentally. Another interaction, called *weak*, that governs radioactive processes and decays, was integrated. A combined description of the electromagnetic and weak interactions was developed in the form of the unified *electroweak* interaction. In combination, these theories make up the Standard Model (SM) of particle physics. Using the SM allows describing the vast majority of the properties of particle physics. It encompasses three of the four known interactions, and can be used to precisely calculate the processes that the interactions can induce.

The force carriers of the weak interaction were discovered in the late 20th century, while the early 21st century saw the discovery of the missing puzzle piece confirming the electroweak unification methodology, the Higgs boson. This last discovery was made possible by the Large Hadron Collider (LHC), a particle collider of 27 km circumference, capable of producing proton-proton collisions at unprecedented center-of-mass energies of up to 13 TeV. After the discovery of the Higgs boson, the effort to map out and measure the Standard Model was reinforced. Any discrepancy that might be found would indicate an inconsistency in this understanding of particle physics.

Even though the Standard Model has thus far been able to describe all phenomena observed in particle collisions with dazzling precision, there are indications that it remains an incomplete description. Circumstances like an apparent 95 % of matter being invisible, oscillations of supposedly massless *neutrino* particles or highly unsatisfactory properties of the Standard Model itself support the notion of physics *beyond the Standard Model*. Consequently, the LHC is also used to search for evidence supporting this kind of new physics. There is no shortage of theories extending the Standard Model, among them Supersymmetry (SUSY).

It uses an additional symmetry, which is central concept in QFT, to try to reconcile certain issues with the Standard Model.

Particle collisions are exploited for research by studying the particles produced in the interactions caused by the collisions. To this end, particle detectors are constructed that serve this exact purpose. At the LHC, four main particle detectors were assembled and have been recording measurements of the collisions for around a decade. While some detectors are designed with specific signatures in mind, others are built for general purpose measurements, such as the ATLAS experiment. ATLAS is able to detect traces of some of the particles that are the result of particle collisions, depending on their properties. This is achieved by using a wide array of different technologies. Some subsystems are used for specific particle types, while others are dedicated to the measurement of specific quantities. Examples are the muon system, and a set of calorimeters, performing energy measurements of leptons, photons and hadrons. Finally, a special detector is located at the very center of the device, which measures the trajectories of any charged particle passing through it. Using sensor elements that are sensitive to particles intersecting them, a series of measurements along the particle trajectory is obtained. With knowledge of the location and orientation of each sensor, the particle trajectory, or *track*, can be recovered. Complex algorithms are deployed for this purpose, and are able to measure particle track properties at highest precision. The complexity of track reconstruction heavily hinges on the activity of an event. In order to fit within available resources, increases in the instantaneous luminosity that are foreseen for the HL-LHC pose challenges for track reconstruction, and will require further improvements.

This thesis presents work that was conducted in two main areas.

Firstly, it reports on improvements and advancements of the software used for track reconstruction. This effort is embedded in the ACTS project that aims to modernize the ATLAS software, while at the same time making it available to be used in other experiments. Aside from this, being an experiment-independent software toolkit, it supports collaboration in the track reconstruction community. ACTS components can be part of a solution to overcome the challenges posed by future collider environments. With a focus on performance and optimal exploitation of available resources, they can help remaining within computing budgets. Therefore, leveraging components developed and improved in this fashion in ATLAS is foreseen, and is an additional goal of the project.

Secondly, a concrete application of track reconstruction is described. An analysis of ATLAS data searching for a specific signature is presented. Different models can result in such a signature, while this thesis uses a specific SUSY model as a reference. In models of this kind, hypothetic long-lived charged particles leave the primary interaction area and travel through the ATLAS tracking system. After about $\mathcal{O}(\mathrm{ns})$, they decay into largely invisible products which leave the detector unnoticed. The initial charged particle will have left measurements in the innermost part of the tracker, the Pixel detector. Dedicated track reconstruction algorithms are deployed to reconstruct these very short tracks.

Three parts form the basic structure of this thesis:

Part I gives an introduction into the backgrounds that are useful for the subsequent parts. A discussion of theoretical aspects of particle physics is given, as is an overview of the ATLAS experiment, whose data is used later on. It also contains a thorough introduction of the concepts used in track reconstruction. Another chapter outlines the challenges of track reconstruction in light of the high-luminosity LHC project.

Part II focusses on the ACTS project and its integration into the ATLAS software, and specific contributions made in this context are described. The work presented in this thesis launched the effort to reintegrate ACTS components back into the ATLAS software. Dedicated chapters give details on the geometry description of current and future ATLAS tracking detectors, a new navigation model, improvements made to the Event Data Model and concurrent handling of sensor misalignments.

Finally, part III presents a search for disappearing tracks using ATLAS data. For this kind of analysis, a reconstruction approach for very short tracks is used. The analysis presented here uses improvements of the reconstruction technique developed prior to the work for this thesis. With the improvement, the analysis is able to use tracks with only three Pixel measurements, as opposed to the four-measurement tracks used in existing analyses. Finally, expected sensitivity using this new approach is quantified and compared with existing results, while expected signal strength upper limits and corresponding mass exclusion limits are shown.

Part I.

Background of theory and experimental techniques

2. Theoretical background

2.1. The Standard Model of Particle Physics

The Standard Model of Particle Physics (SM) [1–4] can be considered the sum of all theories and experimental observations that make up our most complete model of the world at the particle level. With the exception of gravity, all of the fundamental forces are explained in the set of theories contained in the Standard Model. Chapter 2 features an introduction largely based on [5, 6].

The fundamental origins of the SM trace back to the discovery of the electron in 1897 [7] and it continues past its latest addition, the discovery [8, 9] of the Higgs boson [10–15] in 2008. Its history is deeply intertwined with the development of quantum mechanics, and Quantum Field Theory (QFT). An early milestone was the quantization of electromagnetic radiation in the form of photons and the proposition of the photoelectric effect [16]. Early in the 20th century, it was discovered that atoms had substructure, in the form of nuclei, which are in turn composed of *protons* and *neutrons*. In the 1930s, considerations concerning binding forces of the nuclei led to the proposition of a *strong force* [17] that ties them together. Attempts at the development of relativistic quantum mechanics led to the concept of anti-particles [18], which was experimentally confirmed [19] shortly after. In addition, the proposal of the *neutrino*[1] [20] and discovery [21] some 20 years later, supported the mounting effort to thoroughly explore particle physics. In parallel, a QFT describing the *weak force* [22], governing phenomena like radioactive decays, was proposed.

The 1950s and 1960s saw a flurry of discoveries of new particles, and theoretical interpretation ([23–27] among others), which turned out to be a zoo of composites, made up of more elementary *quarks* [28, 29]. These discoveries were largely enabled by the advent of particle colliders, enabling the production of observable decay events.

Advances in quantum field theory led to the formulation of Quantum Electrodynamics (QED) [30–35] in the 1940s, and a similar theory for the strong interaction, Quantum Chromodynamics (QCD) [27, 36] in the following decades.

Initial assumptions of completeness of particle theory with two *leptons*, *electrons* and *muons*, and three quarks, *up*, *down* and *strange*, were disrupted by the discoveries of a third lepton [37] called *tau*, and a fourth quark [38], the *charm*, in the 1970s. This was followed by a fifth quark, the *bottom* [39, 40], in the 1980s. A sixth *top* quark, that was expected thereafter, was finally discovered in the 1990s top [41, 42].

The SM saw its preliminary completion with the development of the electroweak theory.

[1] It has since been understood to have been the electron neutrino.

It unified [1–3] the weak interaction with QED as the *electroweak theory*. The massive interaction particles predicted by the electroweak theory were discovered in the 1980s [43, 44]. The Higgs mechanism, an essential ingredient for electroweak unification, led to the prediction of the Higgs boson, finally discovered in the 2010s.

In spite of the success of the SM, questions and issues remain in particle physics. A discussion of shortcomings of the SM is found later in this chapter (section 2.2), alongside a description of one particular extension called Supersymmetry (section 2.3).

2.1.1. Particle content

The SM contains a number of particles and describes the interaction between them. Particles with spin $\frac{1}{2}$ are referred to as *fermions*. Within this group, two types of fermions exist. The first type are the *leptons*, which consist of *electrons*, *muons* and *taus*. In addition, for each of these an associated *neutrino* exists. The second type are the *quarks*, which also come in three generations of two quarks each: *up* and *down*, *charm* and *strange* and *top* and *bottom*. Both leptons and quarks have an associated quantum number called *lepton number* L_f and *baryon number* B. Leptons carry $L_f = 1$, where f refers to the generation of the lepton, while quarks carry baryon number $\frac{1}{3}$.

The special *natural* unit system is frequently used in High Energy Physics (HEP), which sets the speed of light c and the reduced Planck constant $\hbar$ to 1. As it makes the expression of energies, masses and momenta more convenient, it is used throughout this thesis for quantities related to particles.

All the above particles, with the exception of the neutrinos, carry an electromagnetic charge, and are therefore subjected to the electromagnetic force. This force is mediated by the photon, which itself does not have a charge. Each lepton and quark has a corresponding anti-particle, with identical properties, except for an opposite sign electromagnetic charge and lepton or baryon number, respectively. The quarks and anti-quarks carry fractional charge $\mp\frac{1}{3}$ and $\mp\frac{2}{3}$, while the leptons and anti-leptons carry integer charges -1 and $+1$ in units of the elementary charge e, respectively. Throughout the rest of this thesis, the terms ”lepton”[2], ”neutrino” and ”quark” will generally also refer to the corresponding anti-particles, where not indicated otherwise.

Quarks appear in bound states called *hadrons* in the form of pairs (*mesons*) or triplets (*baryons*)[3]. The *strong force* binds these hadrons together by the exchange of gluons, and couples to the *color-charge* of the quarks. The gluon itself is also color-charged, and therefore also couples to itself. Hadrons are color-neutral, as their constituent quark's charges neutralize. As quarks are color-charged in isolation, they are subjected to an effect called *confinement*. Color-neutral systems resist separation as the strong force increases with larger distances. At some distance, the creation of additional quark-anti-quark pairs from the vacuum is energetically preferable. The resulting system of quarks is then again color-neutral.

[2] ”Electron”, ”muon” and ”tau” will refer to their respective anti-particles as well.
[3] The baryon number is defined such that a baryon has $B = 1$.
[4] Technically, the mass limit is on $\bar{\nu}_e$, and masses can be different in certain theoretical scenarios.

name	charge [e]	spin	mass [MeV]
	quarks		
up	$+2/3$	$1/2$	2.16
down	$-1/3$	$1/2$	4.67
charm	$+2/3$	$1/2$	1.27×10^3
strange	$-1/3$	$1/2$	93
top	$+2/3$	$1/2$	172.76×10^3
bottom	$-1/3$	$1/2$	4.18×10^3
	leptons		
electron	-1	$1/2$	0.51
electron neutrino[4]	0	$1/2$	$< 1.1 \times 10^{-6}$ [45]
muon	-1	$1/2$	105.66
muon neutrino	0	$1/2$	< 0.19
tau	-1	$1/2$	1776.86
tau neutrino	0	$1/2$	< 18.2
	bosons		
gluon	0	1	0
photon	0	1	0
Z	0	1	91.19×10^3
W	± 1	1	80.38×10^3
H	0	0	125.10×10^3

Table 2.1.: Properties of the fermions and bosons. The values are taken from [46] where not indicated otherwise.

All fermions participate in the *weak interaction*. The weak interaction is mediated by a pair of charged particles $W^{\pm}$ and the neutral Z^0. This interaction is responsible for the decay modes of the heavy leptons, muons and taus, which can decay to a lighter lepton and associated neutrino, in addition to a correspondingly charged $W^{\pm}$. Quark decays to a $W^{\pm}$ and another quark are also possible.

The force-mediating particles mentioned above are referred to as *bosons*, and all carry integer spin 1. While the gluon and photon are massless, the $W^{\pm}$ and Z^0 are massive. Their mass is caused by the *Higgs mechanism*, which also results in another neutral spin-0 boson called *Higgs*. Details on all the particles mentioned above are given in table 2.1.

A quantitative mathematical description of the properties and interactions is given in the SM. *Spinor* fields are used to describe fermions, which encode their spins and other properties.

2.1.2. Mathematical structure

This section provides some insight into the mathematical structure of the SM, which uses the mechanism of QFT, and is based on [6, 47]. This kind of theory puts quantized fields $\phi_i(x_\mu)$ as a function of a four-position vector x_μ at its center. The properties and dynamics of the field are defined by a *Lagrangian* density $\mathcal{L}$. In this approach, excitations of these fields can be identified with the particles observed in nature.

Free field equations

QFT takes inspiration from the Euler-Lagrange formalism, which is used in classical mechanics. When applied to the Lagrangian density in the form of the Euler-Lagrange equation

$$\partial_\mu \left(\frac{\partial \mathcal{L}}{\partial(\partial_\mu \phi_i)} \right) - \frac{\partial \mathcal{L}}{\partial \phi_i} = 0, \tag{2.1}$$

field equations can be obtained that govern the evolution of the quantized fields. The field equations exhibit characteristic behavior depending on the spin of the field.

In case of a field A_μ with spin-1 , the Lagrangian without any external fields reads

$$\mathcal{L} = -\frac{1}{16\pi} F^{\mu\nu} F_{\mu\nu} + \frac{1}{8\pi} m^2 A^\nu A_\nu \tag{2.2}$$

where

$$F^{\mu\nu} = \partial^\mu A^\nu - \partial^\nu A^\mu \tag{2.3}$$

is the field tensor and m is the mass of the associated particle. Under application of equation (2.1), the field equation

$$\partial_\mu F^{\mu\nu} + m^2 A^\nu = 0, \tag{2.4}$$

can be obtained.

For spin-$\frac{1}{2}$ fields, the so-called Dirac-Lagrangian is

$$\mathcal{L}_{\text{dir}} = i\bar{\Psi}\gamma^\mu \partial_\mu \Psi - m\bar{\Psi}\Psi. \tag{2.5}$$

Here, Ψ is a Dirac spinor associated with the field, $\bar{\Psi} = \Psi^\dagger \gamma^0$, γ^μ are the Dirac matrices, and m is again the mass of the particle. With equation (2.1) this Lagrangian yields the Dirac equation, and the corresponding adjoint

$$i\gamma^\mu \partial_\mu \Psi - m\Psi = 0 \qquad i\partial_\mu \bar{\Psi}\gamma^\mu + m\bar{\Psi} = 0. \tag{2.6}$$

These two equations determine the field-free evolution of all fermions Ψ and their antiparticles $\bar{\Psi}$.

Local gauge invariance

A critical concept in QFT is local gauge invariance. In general, a Lagrangian is gauge invariant, if the associated gauge transformations do not change the Lagrangian. As an example, the Dirac-Lagrangian from equation (2.5) is gauge invariant under transformations from the $U(1)$ symmetry group. In concrete terms, this means that with a transformation like

$$\Psi \to e^{i\theta}\Psi = \Psi', \tag{2.7}$$

where θ is an arbitrary phase angle,

$$\mathcal{L}_{\text{dir}}(\Psi') = \mathcal{L}_{\text{dir}}(\Psi). \tag{2.8}$$

Since the phase angle θ is the same everywhere, this is referred to as a *global gauge invariance*. However, if the phase angle θ depends on the position x^μ like

$$\Psi \to e^{i\theta(x)}\Psi = e^{-iq\lambda(x)}\Psi, \tag{2.9}$$

this symmetry is broken. $\mathcal{L}_{\text{dir}}$ does not fulfill *local gauge invariance* without modifications.

In order to make the Lagrangian fulfill equation (2.8), it is supplemented with a coupling to a vector *gauge field* A_μ, resulting in

$$\mathcal{L} = i\bar{\Psi}\gamma^\mu\partial_\mu\Psi - m\bar{\Psi}\Psi - (q\bar{\Psi}\gamma^\mu\Psi)A_\mu. \tag{2.10}$$

This gauge field transforms under local gauge transformations like

$$A_\mu \to A_\mu + \partial_\mu\lambda(x). \tag{2.11}$$

This transformation behavior of A_μ can be used to redefine the derivative of Ψ, yielding the *covariant derivative*

$$\mathcal{D}_\mu = \partial_\mu + iqA_\mu. \tag{2.12}$$

Adding equation (2.2) to incorporate the free evolution of A_μ, the resulting Lagrangian is

$$\mathcal{L} = i\bar{\Psi}\gamma^\mu\partial_\mu\Psi - m\bar{\Psi}\Psi - (q\bar{\Psi}\gamma^\mu\Psi)A_\mu - \frac{1}{16\pi}F^{\mu\nu}F_{\mu\nu} + \frac{1}{8\pi}m_A^2A^\nu A_\nu. \tag{2.13}$$

However, the Lagrangian in this form is still not invariant under local $U(1)$ transformation, as $A^\nu A_\nu$ is not. The only way to recover this symmetry is the field A_μ being massless so that $m_A = 0$, and the final term vanishes. In that case, the resulting Lagrangian is the one of QED:

$$\mathcal{L}_{\text{QED}} = \bar{\Psi}(i\gamma^\mu\mathcal{D}_\mu - m)\Psi - \frac{1}{16\pi}F^{\mu\nu}F_{\mu\nu} \tag{2.14}$$

Here, A_μ can be identified with the massless photon field. The first term describes the interactions between Dirac fermions and the photon field, while the latter term is the remaining part of the free Lagrangian of the photon field.

Similarly to what was shown for $\mathcal{L}_{\text{QED}}$, an equivalent Lagrangian can be formulated for the strong interaction. The theory describing this interaction is called Quantum Chromodynamics (QCD). In this case, the symmetry in question is the $SU(3)$ group, which has eight degrees of freedom. As such, eight vector fields are needed to recover local gauge invariance. These eight vector fields can be identified with the gluons, which carry one of eight charge compositions. Finally, the QCD Lagrangian can be written as

$$\mathcal{L}_{\text{QCD}} = \bar{\Psi}(i\gamma^\mu\mathcal{D}^{\text{s}}_\mu - m)\Psi - \frac{1}{16\pi}\mathbf{F}^{\mu\nu}_{\text{s}}\mathbf{F}^{\text{s}}_{\mu\nu}, \tag{2.15}$$

with the strong covariant derivative D_μ^s, and the field tensors $\mathbf{F}_s^{\mu\nu}$ of the gluon fields, which are also massless.

Spontaneous symmetry breaking and the electroweak theory

When trying to formulate a similar Lagrangian that covers the weak interactions, a critical issue is encountered. As opposed to the photon and gluon fields, the required gauge fields of the weak interaction are not massless. As a consequence, simply introducing covariant derivatives results in a Lagrangian that is not invariant under the associated symmetry. With the *Higgs mechanism* [10–15] and the concept of spontaneous symmetry breaking this can be remedied. This section gives an example of spontaneous symmetry breaking using a simplified but unphysical example, and then presents findings when applying the same mechanism to the full theory.

The general idea is introducing an additional field ϕ, which contributes to Lagrangians in the form of a potential $\mathcal{U}(\phi)$. This field can couple to other fields, and break gauge symmetries in the process if $\phi \neq 0$. Depending on the potential $\mathcal{U}(\phi)$, the ground state, that is the field configuration of lowest energy, can either be at $\phi = 0$ or $\phi \neq 0$. In the latter case, symmetries are broken at the ground state, while they can remain intact at higher energies.

Which impact the presence of such a field and potential has on a Lagrangian can be illustrated with a Lagrangian locally invariant under $U(1)$, as in the case of the electromagnetic Lagrangian. Consider a single complex scalar field ϕ and a potential of the form

$$\mathcal{U}(\phi) = \mu^2(\phi^*\phi) + \lambda(\phi^*\phi)^2. \tag{2.16}$$

A Lagrangian for ϕ demanding local gauge invariance results in a gauge field A_μ, equivalent to what was shown before:

$$\mathcal{L} = (\mathcal{D}_\mu\phi)^*(\mathcal{D}_\mu\phi) - \mu^2(\phi^*\phi) + \lambda(\phi^*\phi)^2 - \frac{1}{4}F_{\mu\nu}F^{\mu\nu} \tag{2.17}$$

In this case the vacuum ground state caused by $\mathcal{U}(\phi)$ depends on the choice of parameters μ and λ. If $\mu^2 < 0$, the ground state is found at

$$v = \sqrt{\frac{-\mu^2}{2\lambda}} \neq 0, \tag{2.18}$$

and ϕ can be rewritten as

$$\phi = v + \frac{1}{\sqrt{2}}(\phi_1 + i\phi_2) \tag{2.19}$$

Inserting this field into equation (2.17) leads to

$$\begin{aligned}\mathcal{L} = &-\frac{1}{4}F_{\mu\nu}F^{\mu\nu} + q^2v^2A_\mu A^\mu + \frac{1}{2}(\partial_\mu\phi_1)^2 + \frac{1}{2}(\partial_\mu\phi_2)^2 \\ &-2\lambda v^2\phi_1^2 + \sqrt{2}qvA^\mu\partial_\mu\phi_2 \\ &+\text{cubic} + \text{quartic terms}.\end{aligned} \tag{2.20}$$

Note that the mass term $q^2v^2A_\mu A^\mu$ for the field A_μ emerges as a consequence of its coupling to ϕ. This means that simply because the vacuum has a ground state in which the ϕ field does not vanish, a mass term for the gauge boson is generated.

In total, this leaves two massive fields A_μ and ϕ_1, while ϕ_2 is not massive. Since the potential $\mathcal{U}(\phi)$ is continuous, (ϕ_1, ϕ_2) can be rotated while maintaining the value v. This symmetry breaking is *spontaneous*, because nature arbitrarily chooses which concrete combination of (ϕ_1, ϕ_2) is realized. The fact that ϕ_2 couples in an unphysical way to A_μ motivates a combination where ϕ_2 vanishes, which is a perfectly valid configuration. In the mechanism, the vanishing field ϕ_2 is named a *Goldstone* boson. A number of these Goldstone bosons always appear in the course of spontaneous symmetry breaking, and their number depends on the symmetry. With it removed, a massive gauge boson remains, and a new massive field is introduced by the mechanism.

The Glashow-Weinberg-Salam model

Using the mechanism and ingredients from above, the *Glashow-Weinberg-Salam* [1–3] model allows for a unified theory of electromagnetic and weak interactions. The starting point is the Dirac Lagrangian with spinor fields. The local gauge symmetry under transformations of the $SU(2) \oplus U(1)$ group is demanded. This results in three gauge fields W_μ^i from $SU(2)$ and one X_μ from ($U(1)$). The next step is the inclusion of a doublet of complex scalar fields

$$\phi = \begin{pmatrix} \phi^+ \\ \phi^0 \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} \phi_3 + i\phi_4 \\ \phi_1 + i\phi_2 \end{pmatrix} \tag{2.21}$$

which is called the *Higgs* field. Again, a local gauge transformation can be carefully chosen where unphysical couplings are removed, yielding

$$\phi(x) = \frac{1}{\sqrt{2}} \begin{pmatrix} 0 \\ \sqrt{2}\eta + H(x) \end{pmatrix}. \tag{2.22}$$

where η is a real number. With this form, the resulting expression for the coupling terms between ϕ and W_μ^i, X_μ reads:

$$\begin{aligned} (\mathcal{D}_\mu\phi)^*(\mathcal{D}_\mu\phi) =& \frac{1}{2}(\partial_\mu H)^2 + \frac{g^2\eta^2}{4}\left[(W_\mu^1)^2 + (W_\mu^2)^2\right] \\ &+ \frac{\eta^2}{4}(gW_\mu^3 - g'X_\mu)^2 \\ &+ \text{cubic} + \text{quartic terms}, \end{aligned} \tag{2.23}$$

with coupling constants g and g'.In a similar way as shown before, the Higgs field has introduced mass terms for the four gauge bosons.

In order to arrive at a massless field A_μ, a rotation between W_μ^3 and X_μ by the *Weinberg*

angle θ_W is introduced:

$$\begin{pmatrix} Z_\mu \\ A_\mu \end{pmatrix} = \begin{pmatrix} \cos\theta_W & -\sin\theta_W \\ \sin\theta_W & \cos\theta_W \end{pmatrix} \begin{pmatrix} W_\mu^3 \\ X_\mu \end{pmatrix}. \tag{2.24}$$

This results in mass terms for W_μ^1, W_μ^2 and Z_μ like

$$M_{W_1}^2 = M_{W_2}^2 = M_W^2 = \frac{g^2\eta^2}{2} \qquad \text{and} \qquad M_Z^2 = \frac{M_W^2}{\cos^2\theta_W}. \tag{2.25}$$

A complete Lagrangian needs to additionally include coupling terms between the Higgs field and the fermions. Finally, the massive gauge bosons can be identified with the SM $W^\pm$ and Z_0 bosons, while the massless gauge boson can be identified with the photon.

Similar to the unphysical example, a number of Goldstone fields disappear, while a scalar field remains. Including all their relevant coupling terms results in the Lagrangian for the electroweak interaction. In the case of electroweak symmetry breaking, three Goldstone bosons vanish, while the scalar field $H(x)$ manifests itself as the Higgs boson. It was independently discovered at the Large Hadron Collider (LHC) by the CMS [8] and ATLAS [9] experiments in 2012.

2.1.3. Quantities of interest in collider events

For experimental particle physics, there are a number of quantities of interest that can be defined for particle interactions. These observables are necessary to perform tests of the SM. In collider experiments, the primary observable is the probability for a specific process to occur. It can be measured by counting the number of events N in which a specific process takes place. Dividing this number by the time interval that it was measured in, yields the event rate of the process. By comparing the rate of a specific process with the total interaction rate, the probability for the process can be calculated. This probability is called *cross-section* σ, for historical reasons. In mathematical terms, the event rate and cross-section are related like

$$\frac{\mathrm{d}N}{\mathrm{d}t} = \sigma\mathcal{L} \tag{2.26}$$

via the *luminosity* $\mathcal{L}$ under the assumption of full acceptance and detection efficiency. If complete acceptance is not given, but only a fraction $\mathrm{d}\Omega$ of the full solid angle is accepted, a *differential* rate

$$\frac{\mathrm{d}^2N}{\mathrm{d}t\,\mathrm{d}\Omega} = \frac{\mathrm{d}\sigma}{\mathrm{d}\Omega}\cdot\mathcal{L} \tag{2.27}$$

is defined using the differential cross-section $\frac{\mathrm{d}\sigma}{\mathrm{d}\Omega}$. The cross-section can be calculated from

theory following *Fermi's golden rule* for a specific scattering process like

$$\begin{aligned}\sigma =& \frac{S}{4\sqrt{(p_1 p_2)^2 - (m_1 m_2)^2}} \int |\mathcal{M}_{fi}|^2 (2\pi)^4 \delta(p_1 + p_2 - p_3 \cdots - p_n) \\ & \times \prod_{j=3}^{n} 2\pi\delta\left(p_j^2 - m_j^2\right) \theta\left(p_j^0\right) \frac{\mathrm{d}^4 p_j}{(2\pi)^4}\end{aligned} \tag{2.28}$$

for incoming particles 1 and 2, and outgoing particles 3, 4, …, n. It depends on the momenta p_i and the masses m_i of the incoming and outgoing particles. The statistical factor S is needed to avoid double-counting identical particles, if there are no identical particles, $S = 1$. The delta function $\delta(p_1 + p_2 - p_3 \cdots - p_n)$ ensures momentum conservation. $\delta(p_j^2 - m_j^2)$ guarantees that all outgoing particles are on their respective mass shell. $\Theta(p_j^0)$ enforces that outgoing energies are positive. Another ingredient is the *matrix element*

$$\mathcal{M}_{fi} = \langle f, t \to \infty | i, t \to -\infty \rangle. \tag{2.29}$$

This quantity describes the transition between the initial state i at negative infinite time, and the final state f at positive infinite time. It is the main driver of the characteristic of any process like this, and can be calculated from theory.

A method called *Feynman calculus* provides a strategy to perform the calculation. In this strategy, a set of *Feynman rules* describes how to construct a series that approximates the matrix element. Each term can be represented visually as a *Feynman diagram*, which uniquely specified the shape of the term. In these diagrams, lines represent particles, and vertices represent the couplings among them. As an example, four Feynman diagrams are shown in figure 2.1 for the process $e^+e^- \to e^+e^-$. The initial and final state particles can be seen, in addition to internal particles like photons (γ). Each term is associated with one specific transition from i to f. Possible interference effects between the terms must also be taken into account. For every vertex, the associated term gains a factor of the associated coupling found in the Lagrangian density. These coupling factors include the coupling constant of the relevant interaction. Internal particle lines are associated with a propagator that needs to be included as well. The overall matrix element is then the integral over internal momenta of the sum of all diagrams for a given transition, while ensuring four-momentum conservation at each vertex.

Calculating the matrix element is complicated by the fact that the number of terms contributing to each transition is infinite. Only a finite set of *leading-order* terms exist, for example shown in the top row of figure 2.1. However, it is possible to add couplings of additional particles, which introduce more factors of the relevant coupling constants. Such *higher-order* terms can either feature additional internal particle couplings, called *loops*, or initial or final state radiation of particles. Examples of the former are the two diagrams in the lower row of figure 2.1. In the Feynman calculus, based on perturbation theory, these higher-order terms in the series are perturbative corrections, suppressed by their small coupling constants.

Another complication is that, for higher-order loop diagrams, the momentum integral diverges. This divergence can be circumvented in a systematic way [4], however. To this end,

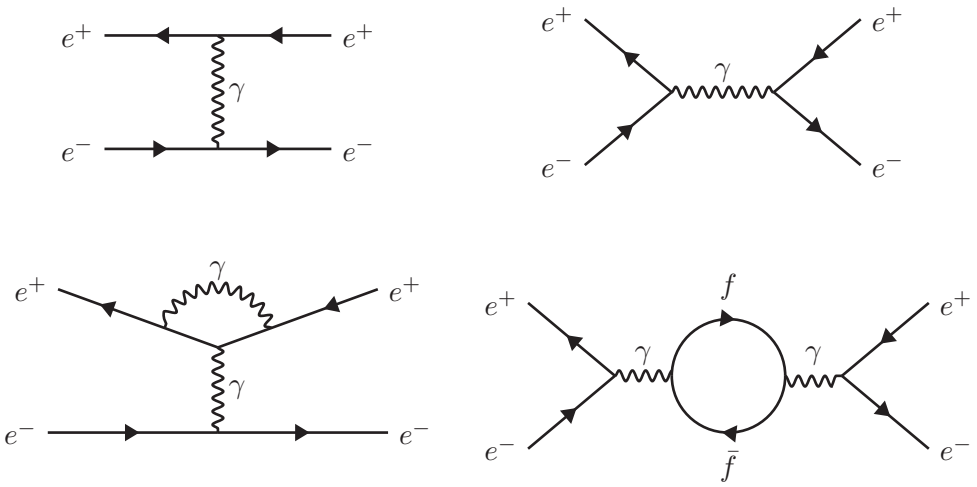


Figure 2.1.: Four Feynman diagrams as examples of configurations contributing to the matrix element for the process $e^+e^- \to e^+e^-$.

the integral is *regularized* using a suitable cutoff. The result is an integral with a finite and a part that diverges in the limit of an infinite cutoff scale. By *renormalizing* the coupling constants and masses that are part of the integral, this divergence can be suppressed: in the limit of an infinite cutoff scale, the integral remains finite.

After renormalization, the coupling constants and masses are now dependent on the energy scale of the process. This means that for higher squared momentum transfer Q^2 of a process, the couplings differ. In QCD, this becomes specifically relevant. In case of large Q^2, the associated coupling constant $\alpha_s = g_s/4\pi$ is small and increases as Q^2 decreases. Therefore, if Q^2 becomes sufficiently small, α_s diverges, and the perturbative approach mentioned before breaks down. As a consequence, the low-energy regime of QCD can generally not be calculated precisely, requiring other approaches and approximation.

This phenomenon is also the cause for the confinement of quarks. With increasing distance between strongly charged quarks, the effective energy scale shrinks. Consequently, the coupling constant increases, until the creation of additional quark-anti-quark pairs is energetically preferable.

2.1.4. Proton-proton interactions

The calculation of transition probabilities that was discussed in the previous section applies in the case of single particles interacting. If the participating particles are composite objects, such as the proton, special care has to be taken to evaluate the probabilities.

Protons consist of three valence quarks: two *up* and one *down* quark. Their combined charge of $2 \times \frac{2}{3} - 1 \times \frac{1}{3}$ explains the overall integer positive charge of $+1$. The proton is a state bound by the strong interaction.

An illustration of a proton is found in figure 2.2. Here, gluons are seen to be exchanged between the three color-charged valence quarks. In addition, the gluons can spontaneously produce additional particle pairs. As gluons are color charged themselves, they can couple

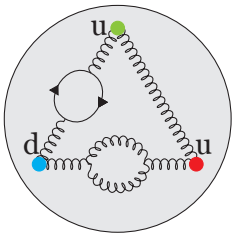


Figure 2.2.: Illustration of a proton and its three valence quarks. Also shown are two intermediate virtual particle pairs produced during gluon exchange.

to other gluons, therefore enabling the gluon loop shown in figure 2.2. It turns out that the presence of virtual gluons is so prevalent in the proton, that over half of its rest energy is continuously carried by these *sea gluons*. Any constituent particle that is contained in the proton is referred to as a *parton*.

In proton-proton collisions, if the center-of-mass energy is large enough, the incoming protons can be considered as incoming bundles of partons, rather than bound composite particles. As a consequence, the fundamental scattering process occurs between individual partons.

Figure 2.3 shows an illustration of two incoming protons interacting with one another. At parton-level a *hard-scattering* process takes place, marked as a red circle. In the example shown in the figure, the parton interaction produces showers of additional partons that are radiated. When a certain energy scale is reached, the parton shower *hadronizes*, resulting in color-neutral hadronic states. As these do not have to be stable, subsequent decays occur, and are shown at the edge of the upper half of figure 2.3 in green. Collections of hadrons that are close together are referred to as *jets*.

Found in the lower half of Figure 2.3 is a representation of an additional secondary interaction, between parts of the proton remnants. Secondary interactions are usually characterized by lower interaction energies. As with the hard-scatter process, a parton shower and subsequent hadronization is shown to occur. The entire proton-proton interaction with all subcomponents is referred of as the *underlying event* of the particle collision.

The total proton momentum is continuously distributed and redistributed among all of the partons. Since the precise evolution of the internal proton structure cannot be calculated from theory, the cross-section calculation is not possible as for elementary particles. To remedy this, the cross-section can be factorized into a cross-section at the parton level, and factors accounting for the structural composition of the proton [49]:

This concept is shown in figure 2.4. Here, two protons interact with each other, and the proton remnants resulting from the interaction are drawn as large arrows. The interaction between partons p_a and p_b of the two protons is visible in the center. The effective final state consists of the particles produced in this parton interaction, in addition to the remnants of the two protons.

$$\sigma_{AB} = \int \mathrm{d}x_a \, \mathrm{d}x_b f_{a/A}(x_a, Q^2) f_{b/B}(x_b, Q^2) \hat{\sigma}_{ab \to X} \tag{2.30}$$

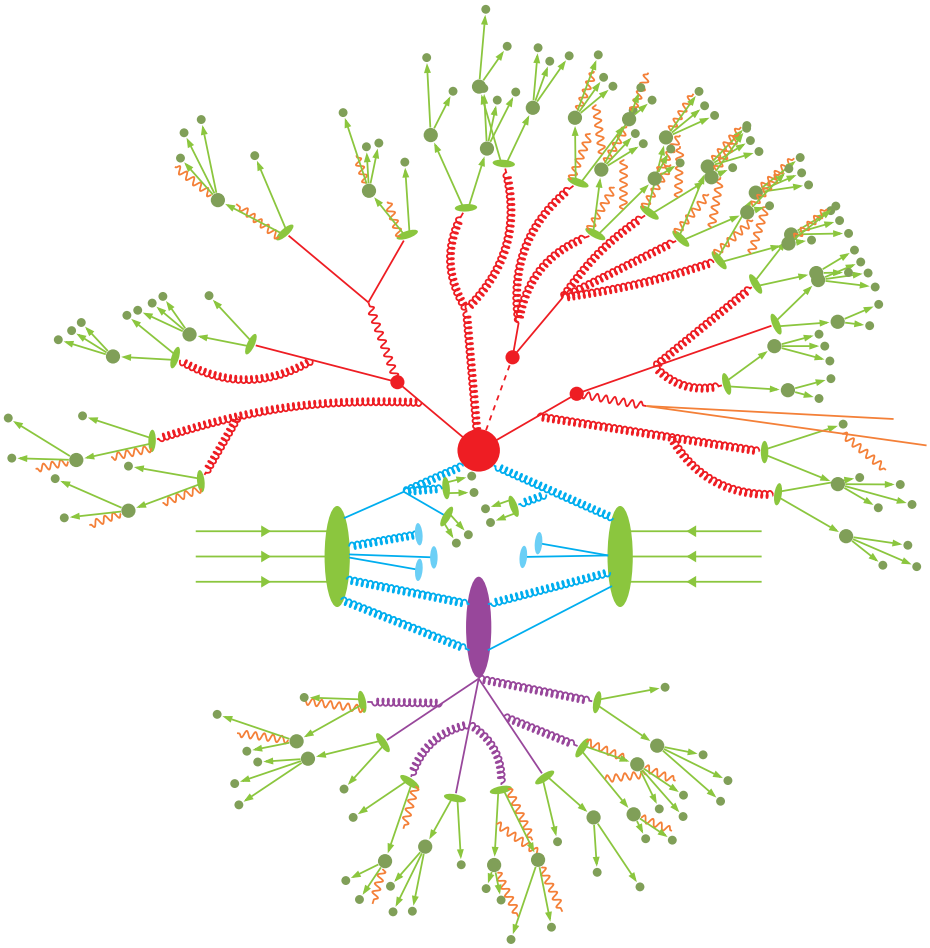


Figure 2.3.: Illustration of the complexity of a proton-proton collision. Two protons are seen entering from the left and right (green ellipses). Two partons undergo the hard-scattering process (red circle), and produce a parton shower, shown in red in the upper half of the image. Partons produced in the shower eventually hadronize and form color-neutral states (small green ellipses). These states further decay into stable particles, seen in the periphery of the top half. The lower half of the image shows a secondary interaction (purple ellipse) of proton remnants. Again, a parton shower results in hadronized states, that decay to stable particles. In addition to these processes, photon radiation is shown to be emitted at every level, drawn in orange. Figure inspired by [48].

The overall cross-section given by an integral over x_a and x_b, which are the fractions of the overall proton momentum carried by partons a, b of protons A and B. The integrand consists of the Parton Distribution Functions (PDFs) $f_{a/A}(x_a, Q^2)$ and $f_{b/B}(x_b, Q^2)$, which describe the probability to find parton a or b of proton A or B at a momentum fraction x_a, x_b of the respective overall proton momentum. This probability also depends on the squared momentum transfer Q^2 of the interaction. $\hat{\sigma}_{ab\to X}$ is the parton-level cross-section for the process $p_a p_b \to X$.

The procedure to derive separation between parton- and proton-level shown above is motivated in the *factorization theorem* of QCD [50]. Even with this factorization, the PDFs are

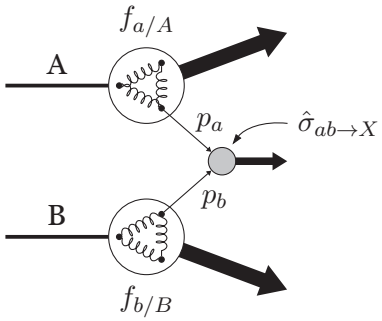


Figure 2.4.: Illustration of an inelastic proton-proton interaction. Two protons enter, and two of their partons, p_a and p_b, interact. The proton remnants exit alongside the parton interaction final state particles.

still needed as ingredients which have to be measured experimentally. Scattering experiments can be conducted to probe the substructure of the proton, and extract PDFs. Deep inelastic scattering using either fixed-target collisions or electron-proton colliders are examples for such experimental measurements. Another possibility is predicting the cross-section of a process observable in proton-proton collisions using an existing PDF. By measuring the cross-section experimentally, the PDF can be refined and made more precise. In all cases, parametrized functional forms are fitted to the obtained data at the Q^2 scale of the experiment. Extrapolations to a different Q^2 than the ones used for the PDF measurement can be achieved using the DGLAP[5] equations [51–53].

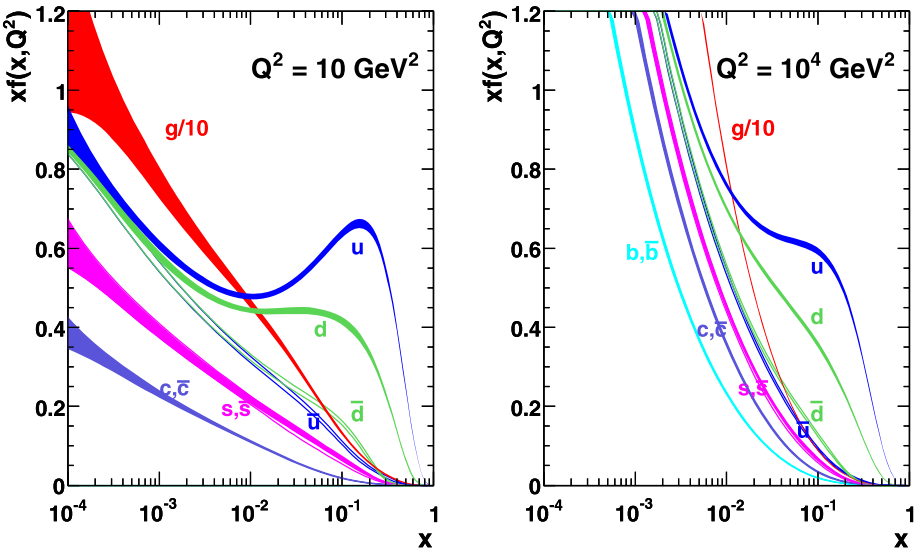


Figure 2.5.: Plot of proton PDFs at two squared momentum transfer values $Q^2 = 10\,\text{GeV}^2$ and $10^4\,\text{GeV}^2$. Shown is the probability to find a certain constituent at a given momentum fraction x. The valence quarks up and down can clearly be seen peaking at large x. Figure taken from [54]

Figure 2.5 contains an example of a set of PDFs for squared momentum transfers of 10 GeV²

[5]Dokshitzer-Gribov-Lipatov-Altarelli-Parisi

and 10^4 GeV². Shown are the distributions for various types of quarks and anti-quarks, as well as gluons. The gluon probability is scaled down by a factor of 10, again stressing its predominance. All PDFs fall toward larger values of x, consistent with the notion that it is exceedingly unlikely to find the proton in a state where a single parton dominates its momentum distribution, while remaining intact.

At $Q^2 = 10\,\text{GeV}^2$, a clear peak of the up quark PDF at about $1/3$ and an excess for the down quark that blends into the falling distribution can be observed. Their dominance in this x range is explained by their continuous presence as the valence quarks of the proton. Their charge conjugated partners $\bar{u}$ and $\bar{d}$ are strongly suppressed in their PDFs, as they can only be found in the proton sea. For charm and strange quark, quark and anti-quarks have identical PDFs, as they can only be found in the sea regardless of their charge.

For larger squared momentum transfer at $Q^2 = 10^4$ GeV² the dominance of u and d is less pronounced, but still visible. At higher momentum fractions x, it is increasingly more likely to resolve sea quarks and gluons, rendering their contributions larger.

A number of collaborations provide PDFs based on deep inelastic scattering, but also incorporating information from LHC measurements. Different approaches to the fitting and extrapolation procedures exist, and are suitable in different applications. Alongside the nominal PDF, a quantification of the uncertainties is provided. These uncertainties cover the original experimental uncertainties, entering the PDF via the data. On the other hand, uncertainties arising from the fitting procedure, and combination with other inputs are also included. Different techniques to calculate and provide this quantification are used. Figure 2.5 indicates the uncertainties with the PDFs displayed in it. They differ both between the various partons, and evolve as a function of both x and Q^2.

2.2. Outstanding issues in the Standard Model

Despite the success of the Standard Model in describing many aspects of particle interactions and properties, it suffers from a number of shortcomings. These issues strongly indicate the existence of *new physics*, augmenting or replacing the SM with a more complete theory beyond it. This section gives a few examples for such outstanding issues. All of them can be addressed in different ways, but as of writing, no observations have lend credibility to any of these extensions of the SM.

The first issue is the lack of unification of the forces. While the electromagnetic and weak interaction were successfully unified with the Glashow-Weinberg-Salam model [1–3] (see section 2.1.2), the same is not true for the strong interaction. Numerous models for *grand unification* exist, but none of them have predicted any observables that could be detected. On the other hand, the fourth fundamental force in nature, gravity, is completely absent in the SM. Proof of a valid QFT formulation of gravity is a milestone on the way to complete unification.

According to the SM neutrinos are exactly massless. A property of the weak interaction that was not discussed in this chapter is that it is maximally parity violating [55]. This means

only particles of left-handed chirality and anti-particles of right-handed chirality participate in the weak interaction, as evident from the Wu experiment [56]. As neutrinos in the SM do not have mass, they do not have right-handed components. This is consistent with the fact that without a right-handed neutrino, no mass term for neutrino be formulated. Due to the observation of neutrino oscillation, where neutrinos change flavor over time, massless neutrinos can unequivocally be ruled out. The SM would therefore have to be supplemented with a mechanism to generate neutrino masses, or right-handed neutrinos would have to exist. The latter would by itself also be an instance of physics beyond the SM.

From the observation of the rotation velocity of galaxies ([57–59] and others), a discrepancy relative to Newtonian mechanics is found. To resolve this tension, either the theory of gravity needs to be modified to account for this observation, or the universe contains large amounts of invisible *dark* matter. Additional hints at dark matter exist, such as the Cosmic Microwave Background (CMB) [60, 61] radiation emitted early in the history of the universe. By studying the CMB, the properties of the baryonic oscillation, taking place at the time of decoupling due to gravitational interaction can be probed via anisotropies. These anisotropies are consistent with the presence of dark matter at this time. All SM particles are ruled out as dark matter candidates. A number of alternative theories exist, a large portion of which follow the Weakly Interacting Massive Particle (WIMP) assumption. One particular dark matter hypothesis from the Supersymmetry theory is discussed in section 2.3.

The Higgs field couples to massive fermions. As a consequence, loop diagrams contribute to the mass parameter of the Higgs field. From experimental observations, this parameter is known to be small, while the loop contributions, especially from the top quark, are very large. The fact that these large contributions would have to cancel each other, to allow the small resulting value, is called the *hierarchy problem* [62–66], and is explored in a little more detail in the following section.

2.3. Supersymmetry

One particular theory beyond the Standard Model is called Supersymmetry (SUSY) [67–72] . Originally devised for more or less aesthetic reasons, it provides a relatively elegant solution to a number of problems of the SM, most notably the hierarchy problem. This section briefly introduces SUSY and some of its characteristics, as well as some of the issues it can solve, and is largely based on [73].

As was discussed in previous sections, the Higgs mechanism explains the masses of the gauge bosons of the weak interaction using a potential that is a function of the Higgs field, and has a non-zero ground state value

$$v = \sqrt{\frac{-\mu^2}{2\lambda}} \neq 0. \tag{2.31}$$

Experimentally, v can be measured to be around $\sim 174\,\text{GeV}$, while the Higgs boson mass is known to be $\sim 125\,\text{GeV}$. As a result, it follows that $\lambda \approx 0.126$, and $\mu^2 \approx -(92.9\,\text{GeV})^2$ in order

to maintain the SM as correct.

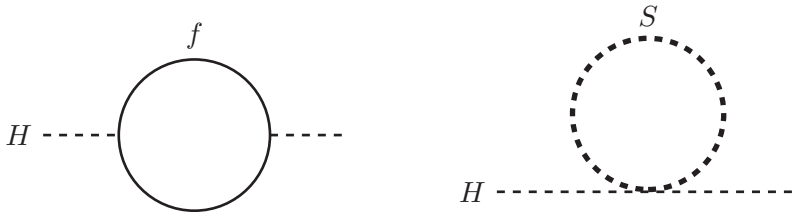


Figure 2.6.: Loop corrections affecting the Higgs field mass parameter μ.

As the complex scalar Higgs field couples to massive particles, loop corrections have an impact on μ. Higgs couplings to fermions and scalar bosons, like the ones shown in figure 2.6, enter the mass correction $\Delta\mu^2$ like

$$\Delta\mu^2 = -\frac{|\lambda_f|^2}{8\pi^2}\Lambda_{\text{UV}}^2 + \ldots + \frac{\lambda_S}{16\pi^2}\left[\Lambda_{\text{UV}}^2 + \ldots\right]. \tag{2.32}$$

These corrections are associated to Lagrangian coupling terms $-\lambda_f H\bar{f}f$ and $\lambda_S|H|^2|S|^2$, and a number of additional terms are represented by the ellipsis. Λ_{UV} is a momentum cutoff scale that regulates the loop integrals.

This turns out to be problematic, as the contributions to $\Delta\mu^2$ due to heavy fermions like the top quark are extremely large compared to the actual value of μ. Since this means that these very large terms would have to miraculously cancel to achieve a very small necessary value of μ, this is unsatisfactory. Taking inspiration from the fact that the contributions from fermions and bosons differ in sign relative to one another, SUSY introduces a way to achieve systematic cancellation of these correction terms. By considering a new *supersymmetry* between fermions and bosons, it generates automatic cancellation terms in $\Delta\mu^2$. The small value of μ is then just an obvious consequence of the symmetry.

An operator is introduced that transforms between fermionic and bosonic states. Additionally, fermions and bosons are paired in *supermultiplets*. Different ways to execute the grouping exist, and the grouped particles are referred to as *superpartners*. All quarks and leptons are paired with new additional scalar bosonic partners, called *scalar fermions* or *sfermions*. The vector gauge bosons of the SM are grouped with new fermionic partners, referred to as *gauginos*. Note that this symmetrization operates on the bosons before electroweak symmetry breaking, such that these supermultiplets are

$$\begin{aligned} g &\leftrightarrow \tilde{g} \\ W^{\pm} &\leftrightarrow \tilde{W}^{\pm} \\ W^0 &\leftrightarrow \tilde{W}^0 \\ B^0 &\leftrightarrow \tilde{B}^0, \end{aligned} \tag{2.33}$$

where the latter three are called *electroweakinos*. To avoid disturbing the electroweak gauge symmetry, two Higgs supermultiplets must be added, one of which contains the SM Higgs. This results in four Higgs scalars H_u^+, H_u^0, H_d^0, H_d^-, and corresponding fermions $\tilde{H}_u^+$, $\tilde{H}_u^0$,

$\tilde{H}_d^0$, $\tilde{H}_d^-$ called *higgsinos*. The exact particle content is variable and depends heavily on the details of the supersymmetric model under study. The Minimal Supersymmetric Standard Model (MSSM) [70–72, 74–76] features the minimal particle spectrum discussed so far. After symmetry breaking, the electroweakinos mix with the higgsinos to form four neutral mass eigenstates called *neutralinos* and two charged ones called *charginos*.

Another very important concept is the introduction of the R-parity [77], defined as

$$P_R = (-1)^{3(B-L)+2s}, \tag{2.34}$$

with the Baryon and Lepton numbers B and L, and the spin s. This quantum number can be required to be conserved to suppress interactions which would violate B and L conservation in isolation. All SUSY particles are assigned $P_R = -1$ and all SM particles receive $P_R = 1$. Depending on the theory, P_R can be conserved or violated to some extent. If it is conserved, it explains the stability of the proton, without having to modify the couplings, which is a primary motivation for the quantity. A consequence is that SUSY particles can only ever be produced in pairs, and the Lightest Supersymmetric Particle (LSP) has to be stable. If the LSP is one of the neutralinos, it poses a promising dark matter candidate. R-parity is defined to be conserved in the MSSM.

Finally, it is apparent that SUSY must be broken at the vacuum. As the superpartners have identical mass to their SM counterparts, if SUSY was unbroken at the ground state, these particles would have been detected thus far. The symmetry breaking must be implemented in a way that retains the required relationship between the various couplings, while still providing a solution to the hierarchy problem in its broken state.

How exactly SUSY breaking works is unclear, and is simply explicitly added to occur spontaneously in the MSSM. This effectively separates the problem of finding an origin of SUSY breaking from the rest of the theory. One possibility is the addition of a hidden sector in which the fundamental symmetry breaking occurs. It is then *communicated* to the visible sector, for instance the MSSM, in some way. There are several candidates for the mediation of symmetry breaking between these two sectors. Plank-scale Mediated Supersymmety Breaking (PMSB) communicates the symmetry breaking via gravity, while Gauge Mediated Supersymmetry Breaking (GMSB) uses the electroweak gauge interactions to execute the mediation. Finally, Anomaly Mediated Supersymmetry Breaking (AMSB) [78] is another approach using an argument related to extra-dimensions and an anomalous violation of a symmetry associated with it.

2.3.1. MSSM and compressed particle mass spectra

Without modification, the absence of any experimental evidence for the newly proposed particles puts pressure on SUSY as a theory. However, with the introduction of a mass parameter μ, the particle spectrum can be split into a high-mass one and a low-mass one, while remaining consistent with positive lepton superpartner masses. The high-mass sector remains out of reach of current collider experiments, but the low-mass sector can potentially

be probed.

Notably contained in this low-mass sector would be the LSP. As mentioned before, the electroweakinos and higgsinos mix. Depending on the mixing, the LSP could feasibly be a nearly pure-higgsino mass eigenstate. The low-mass sector would contain a Dirac fermion $\tilde{\chi}^0_{1,2}$, consisting of the two neutral higgsino fields. On the other hand, a chargino $\tilde{\chi}_1$ of pure-higgsino composition would be found as well.

At the same time, assuming R-parity conservation is given, an LSP higgsino would be a viable Dark Matter (DM) candidate, even though there are cosmological constraints due to DM abundance. If the higgsino mass is too large, the abundance would be larger than observed. A higgsino mass of $\sim 1.1\,\text{TeV}$ [79] is compatible with the observed DM density. On the other hand, there are arguments [80] for a higgsino mass on the order of the electroweak symmetry breaking scale of $\mathcal{O}(100\,\text{GeV})$. This would result in a lower DM abundance from the higgsino, and would require additional DM contributions to fill the gap.

However, this full pure-higgsino scenario is already excluded by direct DM searches [81], as it would result in an excessive higgsino-nucleon scattering cross-section. As a remedy, an effect that occurs at higher energies [79] is needed. In the presence of this effect, the Dirac fermion $\tilde{\chi}^0$ is split into two Majorana fermions $\tilde{\chi}^0_{1,2}$. If the mass difference is $\Delta m_0 > \mathcal{O}(100\,\text{keV})$, the $\tilde{\chi}^0_1 N \to \tilde{\chi}^0_2 N$ scattering rate is sufficiently low [79] to evade exclusion.

Overall, the mass eigenstates $\tilde{\chi}^\pm_1$ and $\tilde{\chi}^0_{1,2}$ are almost mass-degenerate at tree-level, with a small mass-splitting

$$\Delta m_+ = m_{\tilde{\chi}^\pm} - m_{\tilde{\chi}^0} = \Delta m_{\text{tree}} + \Delta m_{\text{rad}} \tag{2.35}$$

that is dominated by Δm_{rad}, while $\Delta m_{\text{tree}} \ll \Delta m_{\text{rad}}$. Electroweak loop corrections cause Δm_{rad}, which therefore drive the observable behavior. The exact value of Δm_{rad} depends on the masses of the chargino and neutralino. Such a nearly degenerate mass-spectrum [82] leads to a dominance of a particular chargino decay: $\tilde{\chi}^\pm_1 \to \tilde{\chi}^0_{1,2}\pi^\pm$. As the pion momentum is driven by the mass splitting, it is very low. Such pions therefore create a characteristic signature of a disappearing charged particle track. A search for this kind of signature with the ATLAS experiment is the focus of part III of this thesis.

3. The ATLAS experiment

The ATLAS[1] experiment [83] at the European Organization for Nuclear Research (CERN) is one of the four major experiments that were planned and realized as part of the Large Hadron Collider (LHC) [84] project. The LHC is capable of producing proton-proton collisions, as well heavy ion collisions in the form of lead-lead or proton-lead beam crossings, at unprecedented center-of-mass energies. The highly energetic beams are produced by a sequence of particle accelerators. At four interaction points, these beams are made to collide with each other. Experiments like ATLAS are constructed around the collision points, and measure the interactions that occur in them. This chapter gives an introduction into the accelerator and the experimental setup of the ATLAS experiment. It loosely follows the description found in [5].

3.1. The Large Hadron Collider

The Large Hadron Collider at CERN in Geneva, Switzerland, is a hadron collider featuring the world-leading center-of-mass energy of $\sqrt{s} = 13\,\text{TeV}$, at the time of writing. Located approximately 100 m underground, it consists of a number of segments, together forming a ring of 27 km in circumference. Bunches of protons and heavy ions travel in opposing directions through two separate beam pipes. Over a thousand dipole magnets force the hadrons on a trajectory along the circumference of the accelerator. In addition, other magnets, including quadrupoles, are used to manage and shape the beam profile, and also focus it for the interactions. As only the proton-proton mode of the LHC is relevant for this thesis, the description of the accelerator focusses on them only.

At four main interactions points, the hadron beams are made to collide with one another. At each of these four locations, one of the major LHC experiments, CMS[2] [85], LHCb[3] [86], ALICE[4] [87] and ATLAS [83], can be found. These experiments consist of detectors designed to measure the interactions products of particle collisions. ALICE is designed for heavy-ion collisions, which the LHC produces in dedicated time periods. The other three experiments are primarily geared toward proton-proton collisions, but can also be used during heavy-ion runs. Proton-proton collisions occur with a frequency of up to 40 MHz, for a time gap between neighboring bunches as low as 25 ns.

The LHC is the final stage in a series of accelerators, where each stage increases the proton

[1] A Toroidal LHC Apparatus
[2] Compact Muon Solenoid
[3] LHC beauty
[4] A Large Ion Collider Experiment

beam energy as the beam travels through them. Initially, protons are extracted and ionized from a hydrogen source, before entering a linear accelerator. At the time of writing, CERN is in the process of migrating this first step from the LINAC2[5] to a newly built LINAC4[6]. This new linear accelerator is capable of a proton output energy of 160 MeV, compared to the 50 MeV beams of the predecessor. Subsequently, protons enter the BOOSTER, where they are accelerated to 1.4 GeV. The next step in the chain is the PS[7], after which protons have gained a total of 26 GeV. The final pre-acceleration stage is the SPS[8], which feeds the LHC with protons at 450 GeV. Protons are injected into the LHC using two transfer lines, one for each direction. In the LHC, the protons reach their final beam energy. Figure 3.1 shows an overview of the accelerator chain, starting from LINAC4 all the way up to the LHC. Each individual (pre-)accelerator is indicated in a different color.

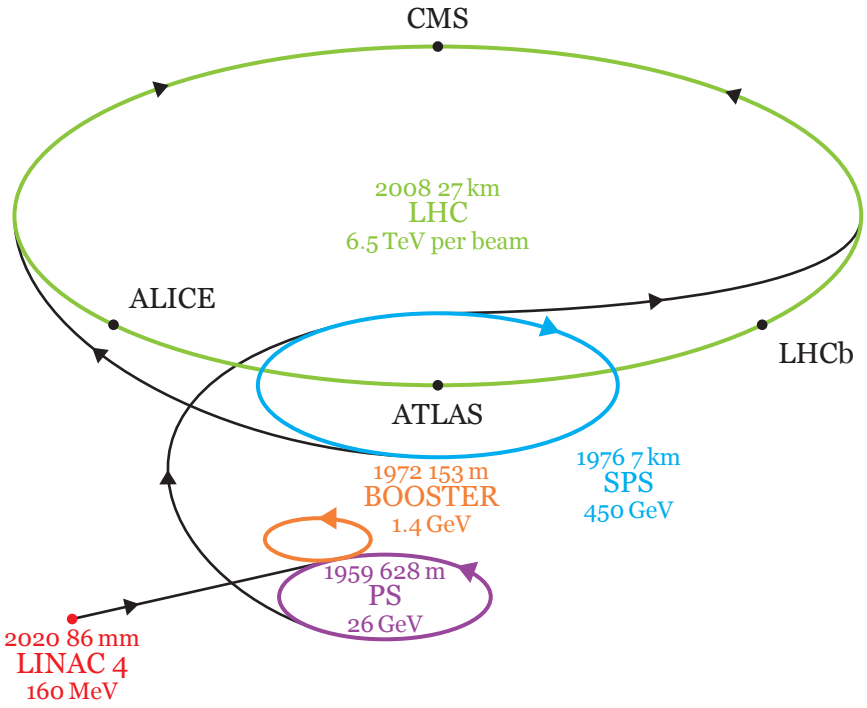


Figure 3.1.: Schematic overview of the LHC accelerator complex at CERN. Traveling through a chain of accelerators, protons are brought up to an energy of 6.5 TeV. Figure inspired by [88].

During the first operating phase of the LHC, Run 1 (2009-2012), each proton beam reached an energy of up to 4 TeV, for a center-of-mass energy of 8 TeV. In Run 2 (2015-2018), the center-of-mass energy was increased to 13 TeV.

One key quantity of interest for colliders is the *luminosity* $\mathcal{L}$ [89], which is a measure of the intensity of the particle beams. Higher luminosity corresponds to an enhanced likelihood for interaction processes to take place in a specific time interval. For colliders, the luminosity

[5]Linear Accelerator 2
[6]Linear Accelerator 4
[7]Proton Synchrotron
[8]Super Proton Synchrotron

can be calculated using

$$\mathcal{L} = \frac{N^2 f N_b}{4\pi\sigma_x\sigma_y}, \tag{3.1}$$

depending on the number of particles N per bunch, the revolution frequency f and the number of proton bunches N_b. σ_x and σ_y describe the size of the overlap between the beams, under the assumption of a Gaussian beam profile. Also, the result has to be corrected for a non-zero crossing angle between the two incoming beams.

The LHC is designed for an instantaneous luminosity of 10^{34} cm^{-2} s^{-1}. Aside from this, the luminosity integrated over a time interval is often quoted as a measure of the delivered amount of collisions. During Run 2, 153 fb^{-1} worth of proton-proton collisions were delivered by the LHC at $\sqrt{s} = 13$ TeV.

3.2. Experimental setup and detector components

The ATLAS experiment is a general purpose particle detector. As such, it is not tailored toward a narrow range of particle phenomena, but tries to be as inclusive as possible in terms of signatures that can be measured. It is constructed symmetrically around one of the four interaction points at the LHC. Geometrically, ATLAS is designed to ensure a large solid angle coverage, to enhance sensitivity to collision products.

Figure 3.2 shows a rendering of the ATLAS detector. The overall structure is 44 m long, and 25 m tall. The picture shows the various subsystems that make up the detector. Major subsystems are the Inner Detector (ID), the calorimeter systems and the Muon Spectrometer (MS). Two magnet systems, a solenoid and a toroid, are also visible. This section introduces these major components in some detail.

3.2.1. Coordinate system

In order to express locations and directions in ATLAS, a specific coordinate system is defined. This frame is a right-handed cartesian reference frame, where the z-axis is aligned with the beam line, along which the protons enter and leave the detector. The y-axis points toward the surface, while the x-axis points toward the center of the LHC ring.

In many cases, the coordinate system is separated into the transverse plane, defined by the x- and y-axes, and the orthogonal longitudinal direction. Two angles are defined: $\phi \in [-\pi, \pi)$ is the azimuth angle in the transverse plane, while $\theta \in [0, \pi]$ is the polar angle along the longitudinal direction. An illustration of these two angles can be found in figure 3.3.

The rapidity

$$y = \frac{1}{2}\ln\left(\frac{E + p_\mathrm{L}}{E - p_\mathrm{L}}\right), \tag{3.2}$$

is a function of the energy E and the longitudinal momentum p_L. It is often used since differences in rapidity are invariant under Lorentz boosts in the longitudinal direction. In

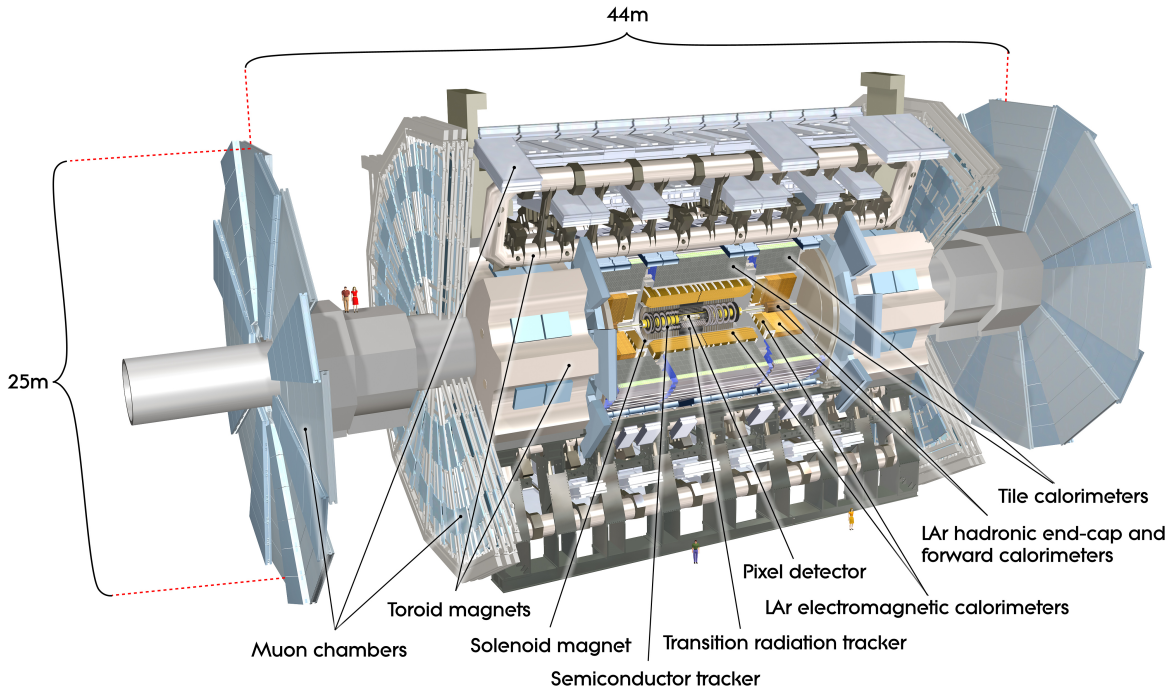


Figure 3.2.: Schematic rendering of the ATLAS detector, showing its overall dimensions. The subsystems that are part of ATLAS are also shown and labeled. Picture taken from [90]

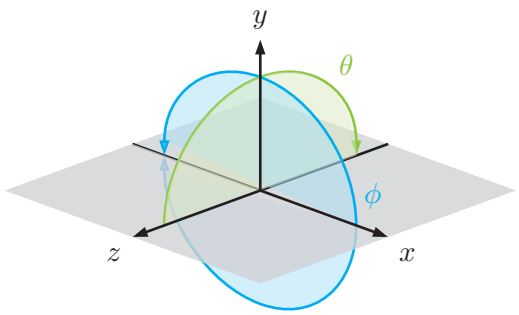


Figure 3.3.: Illustration of the coordinate system used by ATLAS. The transverse and longitudinal angles ϕ and θ are shown.

addition, the *pseudorapidity* η is frequently used, because it can be defined using only the longitudinal angle θ:

$$\eta = -\ln\left[\tan\left(\frac{\theta}{2}\right)\right]. \tag{3.3}$$

For massless particles, $\eta \approx y$. Vanishing η corresponds to $\theta = 90°$, meaning a direction in the xy-plane. Values of $\eta = \pm\infty$ are associated with directions exactly along both directions of the beam line as visible in figure 3.4. As particles *exactly* along the beam line are obviously not measurable by the detector, the fact that their direction cannot be expressed as finite

values of η is acceptable.

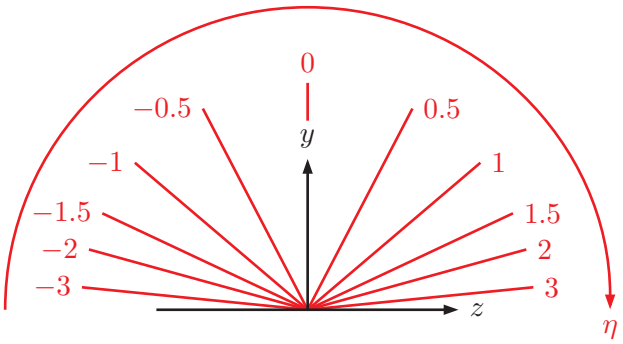


Figure 3.4.: Sketch of the behavior of the pseudorapidity η. $\eta = 0$ corresponds to the direction along the radial axis, while $\eta = \pm\infty$ points along either direction of the beam line.

Finally, two other very important quantities are the *transverse momentum* and *transverse energy*, that can be defined as

$$p_{\mathrm{T}} = \sqrt{p_x^2 + p_y^2} \qquad E_{\mathrm{T}} = \sqrt{{p_{\mathrm{T}}}^2 + m^2}. \tag{3.4}$$

Since only quantities measured in the plane orthogonal to the beam line enter these expressions, they are also invariant under Lorentz boosts along this direction. Apart from this, the detector is only able to measure particles with non-negligible transverse momentum, as the area very close to the beam is not instrumented. As a consequence, the vector sum of all transverse momenta is expected to be balanced, assuming no particles invisible to the detector or outside of its acceptance, and no measurement uncertainties (see section 3.5.4).

3.2.2. Tracking system

The innermost part of the detector, the Inner Detector (ID), is dedicated toward measuring and reconstructing charged particle tracks. Tracks are also used to reconstruct vertices of a given event, which are an ingredient to many higher level reconstruction algorithms. A detailed description of these concepts, and how they are implemented in ATLAS can be found in chapter 4. This section has a focus on the detection systems deployed, and how they serve the purpose of tracking.

As described in detail in [83], the ID consists of three subcomponents: a Pixel detector, a Semiconductor Tracker (SCT), and a Transition Radiation Tracker (TRT). The former two systems use silicon sensors as their detection material. The TRT consists of numerous straw tubes containing a gas mixture and a tungsten wire.

Overall, the ID is subdivided into three regions: one *barrel* region in the center, and two *endcap* regions along the positive and negative direction of the z-axis. Figure 3.5 contains a schematic drawing of these regions of the tracking system. Also shown are examples of a cylindrical sensor layer in the barrel region, and three ring sensor layers in the endcaps. The actual structure and number of barrel and endcap layers differs by ID subsystem.

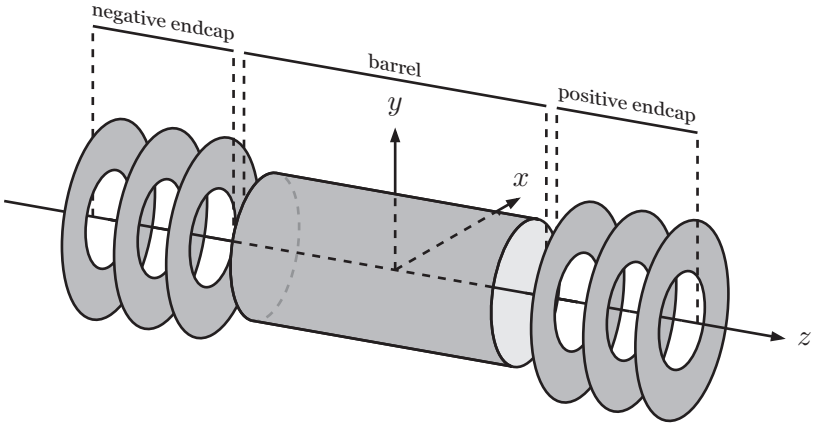


Figure 3.5.: Schematic drawing of how the tracking system is subdivided into a *barrel* region, and two *endcap* regions along the positive and negative direction of the z-axis. The layout shown is exemplary and not a true representation of the ID.

Figure 3.6 shows an overview of the ID systems in the barrel at increasing radii. In the central region between 33.25 mm and 122.5 mm the four barrel layers of the Pixel system are seen. The SCT is located at radii between 299 mm and 514 mm with another four silicon layers in the barrel. At larger radii of 554 mm to 1082 mm the TRT is shown. In the longitudinal direction, the Pixel and SCT detectors cover a range of $|\eta| < 2.5$, while the TRT extends up to slightly over $|\eta| \sim 2.0$.

Track parameter resolution strongly depends on the transverse momentum of the particle and η slice of the tracking system. After the inclusion of the Insertable B-Layer (IBL) [91], the relative momentum resolution can reach values of $p_{\mathrm{T}} \times \sigma(q/p_{\mathrm{T}}) \sim 1\,\%$ depending on η. Absolute resolutions of the impact parameters d_0 and $z_0 \sin\theta$, relative to the primary vertex (see section 4.2), drop as low as $\sigma(d_0) \sim 25\,\mu\mathrm{m}$ and $\sigma(z_0 \sin\theta) \sim 40\,\mu\mathrm{m}$, depending on η and p_{T}. Finally, the spatial resolutions of reconstructed vertices (see section 4.9) reach as low as 8 µm and 24 µm in the transverse and longitudinal direction, respectively.

Pixel detector

The Pixel detector [93–95] forms the central part of the ID. It uses silicon sensors that are segmented in two dimensions. Silicon sensors use semiconducting elements to achieve particle detection. Charged particles passing through the sensors create electron-hole pairs in the depletion zone created by a *pn-junction* between two semiconducting material. As the depletion zone is covered by an electric field, these charges are separated and drift toward the electrodes. Electrons are multiplied as they approach the electrodes, and are then measured. The segmentation allows for the reconstruction of a spatial location.

In total, the Pixel detector consists of four barrel layers in the form of concentric cylinders. The innermost Insertable B-Layer (IBL) [91] was inserted between Run 1 and Run 2 of the LHC. Allowing a measurement at only $r = 33.25\,\mathrm{mm}$, it greatly enhances the spatial resolution of tracks, and consequently of reconstructed vertex positions. Additionally, the Pixel detector

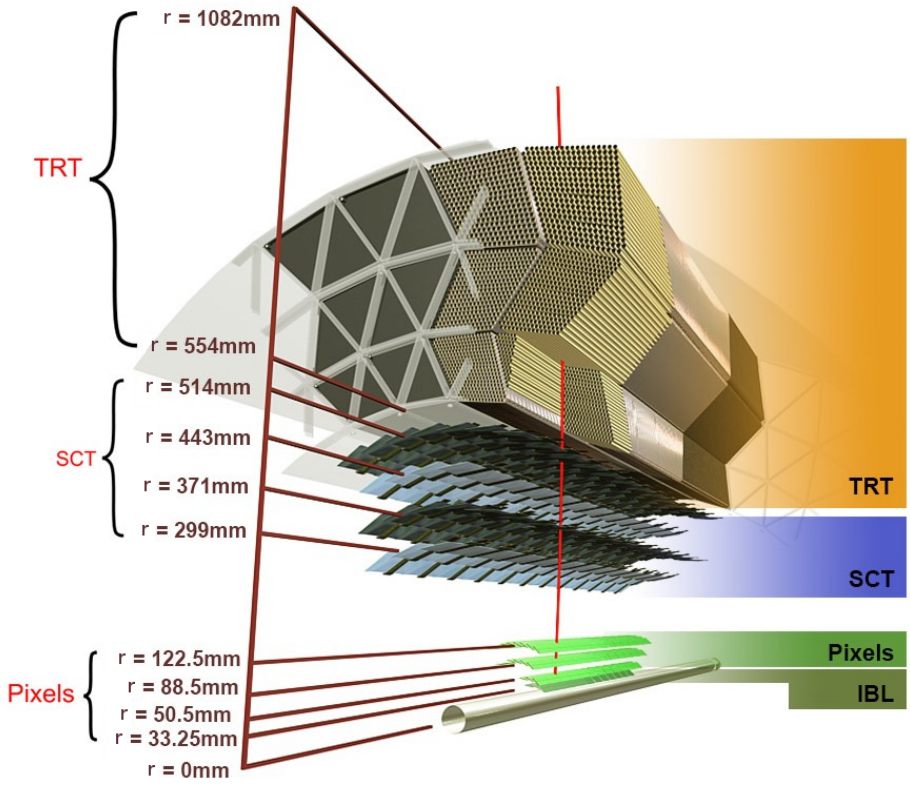


Figure 3.6.: Cross-section of the ATLAS ID in the barrel region. The Pixel, SCT and TRT detectors are visible at increasing radii. Figure taken from [92].

features three endcap layers in the form of disks. On the sensors, individual pixels measure 50 µm by 250 µm for the IBL, and 50 µm by 400 µm in other layers, with some larger pixels around the readout electronics. Corresponding approximate intrinsic accuracies are 9 µm, 60 µm for the IBL [96], and 10 µm, 115 µm for the other layers and disks, in the precision and orthogonal direction.

The readout of the Pixel detector uses the time over a specific threshold to extract a measurement of the amount of energy deposited by a particle in each pixel. This information is used in the reconstruction (see section 4.5) in addition to the list of activated pixels. In total, over 94×10^6 readout channels are spread across 2024 Pixel modules, including the IBL.

SCT detector

Surrounding the Pixel volume, the SCT [94, 95] is located. It consists of four barrel layers featuring pairs of strip modules, rotated by 40 mrad with respect to each other. Since the strip sensors are only capable of measuring particle intersections in one direction, information from these pairs is combined to form a full measurement (see section 4.6). Additionally, the SCT contains nine endcap disks. In the barrel, sensors have a strip pitch of 80 µm in the direction of the bending plane, while the overall sensors measure about 120 mm in the other direction. In the endcaps, strip pitches are constant in the ϕ direction, and measure

161.5 µrad to 207.0 µrad. Intrinsic spatial accuracies are 17 µm and 580 µm in the precision and orthogonal direction, respectively.

As opposed to the Pixel detector, the SCT readout occurs in a binary form: only a list of which strips fired is recorded, rather than the deposited energy. About 12×10^6 readout channels are distributed across 15912 strip sensors.

TRT detector

The outermost component of the ATLAS ID is the TRT [97]. It consists of around 300000 straw tubes of 4 mm radius, arranged in parallel to the beam line in the barrel, and perpendicular to the beam line in the endcap region. The central TRT barrel structure extends up to about $|\eta| < 1$, while the endcap straw layers on either side cover up to about $|\eta| < 2$.

The TRT operates like a drift chamber. In each tube, a tungsten wire is found. Furthermore, the tube is filled with a gas mixture. Here, the tube and wire serve as the electrodes, and an electric potential exists between them. Again, charged particles passing through the straws ionize the ambient gas and thereby create charge pairs, which drift toward the electrodes. When charges approach the central wire, they are amplified, and then detected.

A major objective of the TRT is particle identification. To this end, the straw tubes are surrounded by polymer fibers and foils, facilitating the production of *transition radiation* at material interfaces. Low-energy transition radiation photons are capable of ionizing the TRT gas mixture via photoelectric absorption. As the production of transition radiation depends on the Lorentz factor $\gamma = 1/\sqrt{1 - \left(\frac{v}{c}\right)^2}$, its observation allows conclusions on the mass of the original particle. The raw timing information of the straw tube readout can be converted into an estimate of the radius of closest approach of the incoming particle. Due to uncertainties associated with the drift-time measurement, an intrinsic accuracy of 130 µm is achieved.

3.2.3. Calorimeters

While the innermost part of the detector, the ID, is dedicated toward particle track reconstruction, the systems surrounding it focus on the measurement of particle directions and energies. These *calorimeters* can be grouped into two categories: electromagnetic and hadronic calorimeters.

The general principle of the calorimeters deployed in ATLAS is that of a sampling calorimeter. This type of calorimeter alternates passive and active layers of materials. In the inactive absorber material, particles deposit large amounts of their energy. The active material layers measure the amount of particles that pass through them. Which exact materials are used, depends on specific calorimeter technology. An energy measurement can be extracted due to the way particles interact with the calorimeter material, in particular the absorber material. The main groups of particles of interest are electrons, photons and hadrons.

Electromagnetic showers are created when an electron/positron or photon encounters the calorimeter material. A high-energy incident electron or positron will undergo bremsstrahlung in the vicinity of nuclei, emitting secondary photons. These photons are still energetic enough

to produce electron-positron pairs, with nuclei as recoil partners. The resulting electrons and positrons will again continue to produce bremsstrahlung. A reaction chain of bremsstrahlung and pair production follows, which is called the shower. When shower particles drop below a certain energy threshold, the reactions cease and the shower terminates. Incident photons produce the same reaction chain, by initially undergoing pair production. The shower is characterized by the radiation length X_0, which is a material specific constant that describes the travel distance after which a particle's energy has dropped to $\frac{1}{e}$ of its energy. In combination, the incoming particle energy can be recovered by measuring the total number of particles produced in the shower. More particles correspond to more energy, while the exact relationship needs to be determined in calibration.

Hadrons also produce showers, albeit of a different character. When a hadron passes through dense material, it interacts directly with its nuclei. These inelastic interactions produce additional hadronic particles, largely pions. These secondary hadrons propagate the shower. In case of neutral pions or muons, decays like $\pi^0 \to \gamma\gamma$ or $\mu^\pm \to e^\pm \nu_e/\bar{\nu}_e$ occur. As a consequence hadronic showers are likely to contain electromagnetic subshowers from such decays. In addition, nuclear excitations by the inelastic nuclear interactions can remove energy from the shower. Depending on the material, energy is released back into the shower via nuclear deexcitation. The absorption length λ can be defined in a similar way to the electromagnetic radiation length X_0. Hadronic showers develop more slowly as a function of depth compared to electromagnetic showers.

Electromagnetic calorimeter

In ATLAS, the electromagnetic calorimeter consists of lead as an absorber material, interspersed with liquid argon as the active component. In total, it covers angles up to $|\eta| = 3.2$, with a barrel and endcap component. The geometric arrangement of the active and passive layers is in the form of an accordion shape, which is displayed in figure 3.7.

Readout is performed in cells, which segment the electromagnetic calorimeter both in the longitudinal as well as the depth direction. Three segmentation layers at increasing radii feature differing segmentation in the longitudinal direction, with the innermost layer having the highest granularity. Segmentation in η and ϕ allows a measurement of particle directions, while the segmentation in the depth direction enables a measurement of the shape of showers. A barrel module like the one shown in figure 3.7 features 3424 readout cells, while in the endcaps, modules have 3984 readout channels, for around 1.7×10^5 channels in total. The minimum depth of the electromagnetic calorimeter in units of radiation lengths is $22X_0$.

Following [98], the relative energy resolution for electromagnetic objects can be parametrized like

$$\frac{\Delta E}{E} = \frac{a}{\sqrt{E}} \oplus \frac{b}{E} \oplus c. \tag{3.5}$$

Here, a, b and c are resolution parameters that can be identified with different effects: the sampling term $a/\sqrt{E}$ describes stochastic effects, the b/E term is related to noise and c is

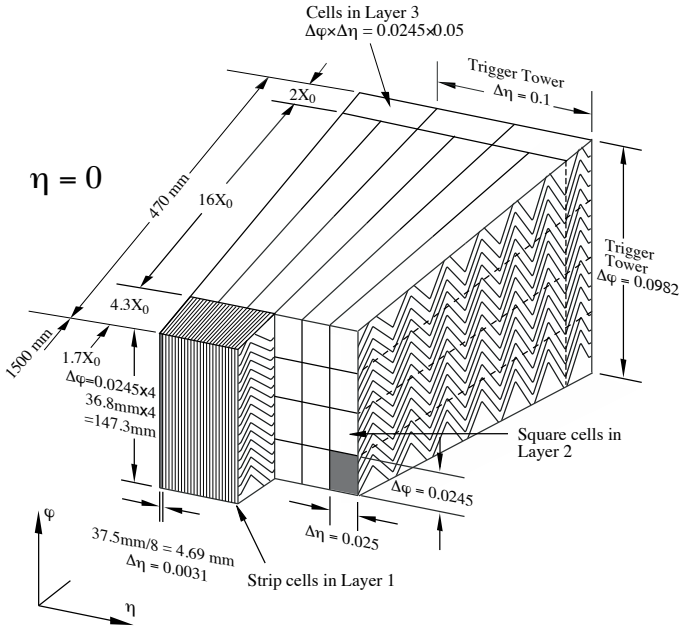


Figure 3.7.: Illustration of the accordion structure of an electromagnetic calorimeter module, in the barrel. The structure is segmented in η and ϕ, as well as the radial direction. Figure taken from [83].

a constant offset. The design values of these parameters are $a \approx 10\,\%$, $b \propto 350 \cosh \eta$ and $c \approx 0.7\,\%$, and were verified in test beam measurements [99].

Hadronic calorimeters

Since hadrons typically reach further into the calorimeter material, the ATLAS hadronic calorimeters surround the electromagnetic calorimeter. Three different components make up the hadronic calorimeters, with differing properties. The design resolution of hadron energy measurements in ATLAS is

$$\frac{\Delta E}{E} = \frac{50\,\%}{\sqrt{E}} \oplus 3\,\%. \tag{3.6}$$

The *tile calorimeter* consists of a central barrel component, and two extended barrels on either side, parallel to the beam axis. In combination, it covers the range of $0 < |\eta| < 1.8$. Steel absorbers are interspersed with scintillator tiles as active material. Along the radial direction, the tile calorimeter measures 7.4λ in units of the absorption length. Overall, the barrel and extended barrel tile calorimeters feature 9852 readout channels. The tile calorimeter pion energy resolution was measured in test beams [100] to be around

$$\frac{\Delta E}{E} = \frac{48.6\,\%}{\sqrt{E}} \oplus 2.46\,\%, \tag{3.7}$$

with some variation depending on the incident angle.

The Hadronic Endcap Calorimeter (HEC) is located at a range of $1.5 < |\eta| < 3.2$, and

extends the reach of hadronic calorimetry further to the beam line. Liquid argon is used as the active material, as is the case in the electromagnetic calorimeters, while copper absorbers are used here. An overall thickness of around 10λ is found [99]. Readout is performed using 5632 channels connected to the HEC. In test beam measurements [101], the energy resolution of the HEC was found to be

$$\frac{\Delta E}{E} = \frac{21.4\,\%}{\sqrt{E}} \oplus 0.3\,\% \qquad \text{and} \qquad \frac{\Delta E}{E} = \frac{70.6\,\%}{\sqrt{E}} \oplus 5.8\,\%, \tag{3.8}$$

for electrons/positrons and pions, respectively.

Forward calorimeter

Finally, the Forward Calorimeter (FCal) covers a range of $3.1 < |\eta| < 4.9$, and is used to provide both electromagnetic and hadronic calorimetry. To this end, the FCal is composed of three types of modules along the depth direction, where the innermost modules are geared toward electromagnetic energy measurement. Again, active liquid argon elements are used, in addition to copper as absorbers for the innermost modules and tungsten for the outer ones. In combination, the FCal has a thickness of $208.1X_0$ and 9.94λ along the depth direction in units of the radiation and absorption lengths, respectively. A total of 1762 readout channels are split across the electromagnetic and hadronic modules of the FCal. Electron/positron and pion energy resolutions were measured in test beams to be about

$$\frac{\Delta E}{E} = \frac{28.5\,\%}{\sqrt{E}} \oplus 3.5\,\% \qquad \text{and} \qquad \frac{\Delta E}{E} = \frac{94\,\%}{\sqrt{E}} \oplus 7.5\,\%. \tag{3.9}$$

3.2.4. Muon system

For the detection of muons, ATLAS uses a dedicated Muon Spectrometer (MS) [102] in addition to the ID. It surrounds both the ID and the calorimeters. At momenta between $\mathcal{O}(100\,\text{MeV})$ and $\mathcal{O}(100\,\text{GeV})$, muons do not strongly ionize material they traverse. This covers a large fraction of muon momenta typically seen in ATLAS. Therefore, muons are expected to pass through the inner subsystems with negligible energy loss. While muons leave traces in the tracking detectors, and can be reconstructed using the ID, the addition of the MS enables more precise measurements of muon quantities and more precise identification. A dedicated toroid magnet system covers the MS with enough magnetic field to observe additional bending. The target relative momentum resolution for the ATLAS MS is 10 % for muons at 1 TeV [83].

The MS uses a number of Monitored Drift Tube (MDT) chambers to measure the curvature of tracks along the bending direction of the toroid field. At larger pseudorapidities, Cathode Strip Chambers (CSCs) can be found that feature higher granularity. Additionally, the MS features hardware designed to allow triggering on muon signatures within $|\eta| < 2.4$. To this end, Resistive Plate Chambers (RPCs) are used in the barrel, while Thin Gap Chambers (TGCs) are used in the endcap regions. Parts of the MS trigger systems are being upgraded with a

New Small Wheel (NSW) [103] at the time of writing, which will enable better trigger level rejection of muon signals not originating from the interaction region.

The over a thousand MDT chambers feature an intrinsic resolution of about 35 µm in the longitudinal direction per chamber, and over 3.5×10^5 readout channels. CSC intrinsic accuracy is about 40 µm per chamber, and 32 chambers with a combined 3×10^4 readout channels exist. The RPCs and TGCs feature weaker intrinsic resolutions at 10 mm and up to 6 mm. Each of the systems feature well over 3×10^5 readout channels, across about 500 and 3500 chambers, respectively.

3.2.5. Solenoid and toroid magnets

To enable momentum measurements in the ID and the MS by means of track curvature, the detector needs to be submerged in magnetic fields. This is achieved by two superconducting magnet systems: a solenoid [104] surrounding the ID, and a toroid [105, 106] that is located inside the MS, and is split across a barrel and two endcap structures.

The solenoid magnet is capable of producing a highly homogeneous magnetic field of 2 T parallel to the z-axis at the interaction point. This homogeneity is necessary for example for the reconstruction of interaction vertices from particle tracks. On the other hand, the toroid produces a radial magnetic field in the volume of the muon system, reaching about 1 T in its center.

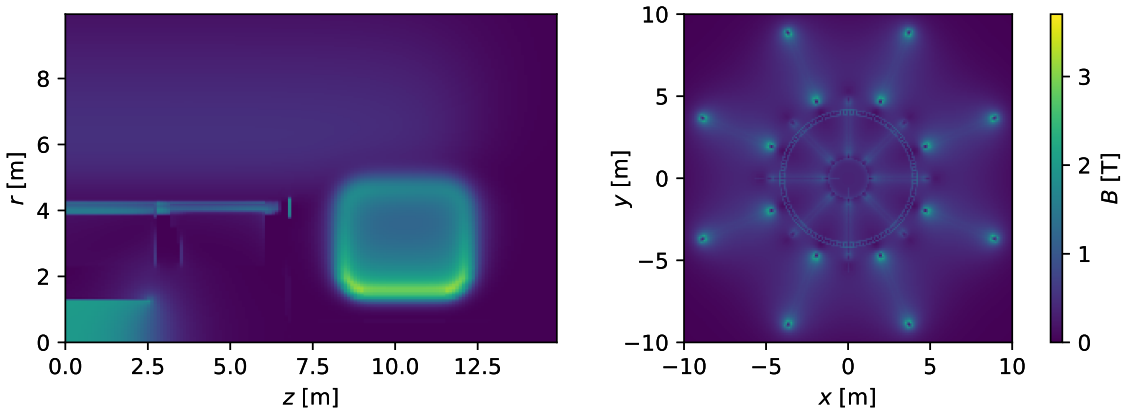


Figure 3.8.: Magnetic field strength in the ATLAS experiment. Shown on the left is the field of the rz-plane, while on the right an average in the xy-plane is seen. The underlying field maps are taken from [107].

Figure 3.8 shows the magnetic field in ATLAS, both in the rz- and the xy-plane. In rz, the solenoid in the ID ($|z| \lesssim 2.5\,\text{m}$, $r \lesssim 1.2\,\text{m}$), and the endcap toroid ($8\,\text{m} \lesssim |z| \lesssim 12.5\,\text{m}$, $1.2\,\text{m} \lesssim r \lesssim 5\,\text{m}$) can be seen clearly. In the central part of the solenoid, the field is strongly homogeneous at 2 T. Between $1.7\,\text{m} < |z| < 1.8\,\text{m}$, the solenoid field drops to $\sim 0.9\,\text{T}$ at the end of the ID volume. The endcap toroid produces a more heterogeneous field, which reaches $> 2\,\text{T}$ at lower radii. The barrel toroid system is only barely visible at larger radii around the

coils. In the xy- view, the barrel toroid coils can be spotted between $\sqrt{x^2 + y^2} \sim 5\,\text{m}$ and 10 m. Within 5 m of the center, the endcap toroid coils can be seen.

3.3. Computing

For the successful operation of the ATLAS experiment, computing plays a critical role. Aspects of computing are used to operate and monitor the ATLAS detector, make decisions about which events to collect by looking at their properties, and recording them to permanent storage. Additionally, reconstruction of physical quantities, calibration, and analyses all fall into the computing domain. It is divided into two categories: online and offline computing, and is described in this section.

3.3.1. Online computing

Online computing describes processes that occur during the operation of the detector, and are directly tied to it, although there is a certain variation of the exact definition. Data taking with the detector requires the steering and control of a large number of hardware systems, related to sensitive elements, but also to auxiliary components like the magnets. All systems need to be configured and monitored to ensure good operation of the detector.

Another important part is the trigger, which serves the purpose of determining events of interest for further analysis, and discarding other events. It is implemented partially in hardware (L1) and in software (HLT). The readout system, controlled by the online software, marshals detector signals from the frontend electronics on the detectors to permanent storage.

Monitoring also occurs at the physics data level: Data quality monitoring during data-taking uses an express readout stream that allows quick execution of reconstruction jobs. While this reconstructed data is highly preliminary, it allows catching problems early on during operation.

The readout system transfers data to the ATLAS Tier-0 facility [108] at CERN. Here, local batch resources are used to promptly reconstruct recorded data. Alongside this, frequent calibration loops are used to determine the conditions that are used for this type of reconstruction. As Tier-0 uses offline software, its usage for prompt reconstruction and the calibration loop sit at the threshold of online and offline computing.

3.3.2. Offline computing

Offline computing refers to all computing that is not immediately connected to the operation of the experiment itself. Athena [109], the ATLAS offline software framework, is used for a large fraction of tasks required. It hosts the reconstruction and analysis software, but is also used for the simulation of particle interactions. All of these topics are described in more detail in later sections. Athena is built on top of the Gaudi [110] event processing framework. It works by executing a number of *algorithm* components for each event. Events can either be read from an input file, or generated, as is needed for simulation. Algorithms

can share information by reading or writing to a shared whiteboard structure. Furthermore, algorithms can use a number of tools, each of which provides specialized functionalities which can be reused across algorithms. These two components are complemented by a set of services, that orchestrate aspects like reading of auxiliary information, file input reading and output writing. The execution model for the event processing sequence is being migrated to a concurrent paradigm, consequences of which are relevant in chapter 5, and section 7.7. Athena is also deployed in the High Level Trigger (HLT) to drive some of the trigger reconstruction algorithms.

As the amount of data recorded is substantial, local batch resources were identified to be insufficient. During the construction of the LHC, ATLAS and other experiments implemented the Worldwide LHC Computing Grid (WLCG) in cooperation with CERN. It is a tiered system of computing centers, that share data processing loads and storage resources. For example, the Tier-0 center distributes the raw and reconstructed data to a number of Tier-1 sites for storage. Additional tiers are part of the grid and used for different purposes.

Another aspect of grid computing are simulation and reprocessing campaigns. Monte Carlo (MC) simulation is executed in a distributed way across the grid sites. Similarly, when a new reconstruction software version is released, already reconstructed data is reprocessed from time to time, to benefit from new developments. Any computing task at this scale is centrally orchestrated and distributed. With the implementation of more aggressive usage of tape storage for permanent archival, such computing campaigns might also require staging inputs to the various execution sites before processing.

Data processing and reconstruction in ATLAS uses a number of different formats [111], each with a distinct purpose. The RAW format stores digitized and preprocessed signals from the detector at about 1 MB per event, and is permanently archived. Reconstruction is executed on the RAW format, and results are written into the ESD[9] format. ESD contains detailed information from the reconstruction, and is therefore suitable for the calculation of calibration parameters. Finally, AOD[10] and its successor xAOD [112] are stripped of the detailed information not immediately necessary for analysis. Target sizes for ESD and (x)AOD are about 500 kB and 100 kB, respectively.

In Run 2, ATLAS employed an additional processing step: the central xAOD is filtered by a *derivation framework* [113] into a number of derived data formats that are each geared toward a specific group of phenomena, or analyses. Event information can also partially be stripped, to remove unneeded information even from events that are of interest. Finally, the derivation can augment the format with additional computed quantities. The last step is often used to attach the output of centrally executed algorithms like calibration information and selection procedures. While the event size of these DAODs[11] is smaller than that of the original xAOD, their large number accounts for a significant portion of ATLAS storage consumption. This DAOD format is the one most commonly used for analysis in ATLAS. The derivation approach and the analysis model is being revised for Run-3, and is outlined in section 5.2.

[9] Event Summary Data
[10] Analysis Object Data
[11] Derived AOD

3.4. Trigger and data acquisition

3.4.1. Trigger system

As described before, proton-proton bunch crossings occur as frequently as every 25 ns in the ATLAS interaction point. However, most of the physics processes that are of interest are exceedingly rare. It is therefore neither feasible nor desirable to fully read out the detector after every bunch crossing, as this would result in excessive amounts of data. To counteract this, a trigger system is deployed. Effectively operating at the bunch crossing rate, it attempts to filter out as many uninteresting events as possible, as early as possible. At the same time, it should be sensitive enough, that bunch crossings which lead to hard-scattering processes of interest are in fact recorded.

The ATLAS trigger system consists of two stages, the Level 1 (L1) trigger and High Level Trigger (HLT). While the L1 operates at the full bunch crossing rate, the HLT only receives inputs when the L1 has accepted an event beforehand. The trigger system is necessarily coupled tightly to the Data Acquisition (DAQ) (see section 3.4.3), as it coordinates when events are actually acquired. A variety of trigger algorithms to classify different types of signatures exist.

The various algorithms from both triggers are combined into a collection of *trigger chains*. Each chain exactly defines the algorithm outputs that are required, and any relevant thresholds and parameters. The collection of chains and configurations is referred to as a *trigger menu*.

Enabled chains are tuned carefully so that their combined output does not exceed the bandwidth of the readout system. Thresholds can be adjusted to lower the acceptance rate of a trigger. In some cases, where this is undesirable, the trigger chain is *prescaled*. This means that only a fixed fraction of trigger accept signals actually result in the event being recorded. Raw events amount to about $\mathcal{O}(1\,\mathrm{MB})$ in size, while the output rate after online selection of events is up to 1 kHz, resulting in a recording data rate of roughly $1\,\mathrm{GB\,s^{-1}}$.

Level 1 trigger

In order to cope with the high frequency, the L1 is entirely implemented in hardware, either using custom designed circuits, or reprogrammable boards. During data-taking, event data is buffered by the frontend electronics. The L1 is able to access event information without requiring a full detector readout. To this end, only coarse-granularity information from the calorimeters and MS is used.

For the calorimeters, analog signals are first digitized, and a preliminary calibration is applied. Then these signals are sent to two subcomponents, dedicated to trigger-level recognition of electrons, photons and taus on one hand, and jet candidates and event-level sums of transverse momenta and missing transverse energies on the other hand. As the missing transverse energy triggers are of importance for the analysis presented in part III of this thesis, they are discussed in more detail in section 3.4.2

L1 electron and photon reconstruction [114] begins by analyzing trigger towers of granularity

0.1×0.1 in η and ϕ. These towers are segmented into electromagnetic and hadronic parts along their pointing direction. Towers belonging to the electromagnetic calorimeters are used. In a sliding window algorithm, the transverse energy is calculated from a pair of towers in the central 2×2 region of the window, which yield the largest energy value. An energy threshold, which depends on the configuration of the L1 algorithm, is applied. Finally, isolation criteria based on the environment of the selected towers are applied either only to electromagnetic signals or to hadronic signals as well. Tau leptons are recognized at L1 using the same trigger towers as above [115]. Energy threshold and isolation requirements are applied in a similar way as for electrons and photons, but also using the hadronic calorimeter information.

The algorithm for recognition of jet signals [116] operates on 2×2 trigger towers as the basic unit. Towers are summed along the depth axis. Windows of varying sizes are used to first identify jet candidates, and then measure the transverse energy of the candidates. Varying thresholds are applied to the jet candidates depending on algorithm configuration.

Triggering on muon signatures uses information from dedicated trigger chambers that are part of the MS [117]. By analyzing observed hit patterns, and comparing them to the expectation for a muon of infinite momentum, the transverse momentum of a muon candidate is estimated. In the barrel, coincidental hits in the RPCs are required. The number of hits depends on the configuration of the algorithm. Multiple instances of the algorithm can be combined with one another to detector signatures with multiple muons. In the endcap region, hit coincidences in the TGC stations are used to identify muons. In the central end of the endcap region, additional signals from the hadronic calorimeter are used to suppress backgrounds.

A Central Trigger Processor (CTP) is responsible for combining the information from the calorimeter and muon L1 triggers. In addition, a topological trigger system exists which can calculate geometric and kinematic combinations of the lower-level trigger objects. These quantities are also used by the CTP to decide whether to issue an event accept.

Overall, the L1 runs at the full bunch crossing rate of up to 40 MHz, and issues event accepts at a maximum of 100 kHz. within a latency of 2.5 µs [118]. Events accepted by the L1 trigger are acquired from the frontend electronics by the readout system, which also buffers the information for the next stage of the trigger, the HLT.

High Level Trigger

Events which have been accepted by the L1 are passed on to the High Level Trigger (HLT). As opposed to the first level, this trigger is implemented in software, and runs entirely on a dedicated Central Processing Unit (CPU) farm, comprising about 40000 processing units. In addition to the L1 information, the HLT is able to access data from the tracking system, as well as additional muon detectors.

HLT algorithms are close to the full reconstruction software found in Athena, and even share code in some cases. One difference is that the HLT is able to reconstruct only parts of an event, which is needed to keep latencies low. The HLT is capable of delivering trigger decisions within about $\mathcal{O}(100\,\text{ms})$.

To this end, it can request partial event data from the readout system buffer. Regions of Interest (RoIs) identified at L1 are handed over to the HLT, which uses these to determine which portions of the event to request. A benefit of the software implementation is that the HLT can more flexibly evaluate physics signatures, and use more sophisticated particle identification algorithms to select objects. As in the L1, various algorithms are available and are scheduled in different configurations. Typically, relatively lenient but fast algorithms are executed first to allow for early rejection in an algorithm chain.

Finally, the HLT issues an accept or reject signal. Events are accepted at up to 1.2 kHz, where about 1 kHz are associated with the main physics output stream. After the HLT decision, the readout system is instructed to either be cleared from the buffer in the readout system, or sent to the Data Acquisition (DAQ) system. The average output rate during Run 2 amounted to about 1.2 GB s^{-1} across the streams.

3.4.2. Missing transverse energy triggers

One particular group of trigger algorithms attempts to reconstruct the missing transverse energy of the event (see section 3.5.4). In the analysis presented in chapter 9, certain $E_{\mathrm{T}}^{\mathrm{miss}}$ trigger chains are used, which combine information from the L1 and the HLT. As described in [119], these trigger chains require a minimum value of the missing transverse momentum over a series of thresholds in successive trigger stages.

The first $E_{\mathrm{T}}^{\mathrm{miss}}$ threshold is applied at L1 on a value that is directly calculated from the vector sum of the energies of the coarse cluster towers. Alternatively, the HLT can also calculate a value for $E_{\mathrm{T}}^{\mathrm{miss}}$ directly from the set of calibrated *topological clusters* (see section 3.5.1) found in the calorimeter. The second and third chain components are executed in the HLT. Multiple algorithms are available in the HLT, which result in different values for $E_{\mathrm{T}}^{\mathrm{miss}}$. A relatively simple algorithm calculates $E_{\mathrm{T}}^{\mathrm{miss}}$ directly from all calorimeter cells whose signal-to-noise ratio is above a certain threshold.

Another algorithm exists that is designed for local pile-up suppression. This algorithm attempts to compensate for calorimeter signals that have low E_{T} and are caused by pile-up interactions. It groups topological clusters (see section 3.5.1) into large patches roughly corresponding to the typical size of a jet signal. It then tries to identify which patches originate from the hard-scatter process, and which stem from pile-up activity, where the assumption is that these feature low E_{T}. A fitting procedure runs to estimate the contribution of similar low-E_{T} signals to the high-E_{T} ones, believed to be associated with the primary interaction. After the contribution is estimated, it is subtracted from the hard-scatter patches, which are then used to calculate an $E_{\mathrm{T}}^{\mathrm{miss}}$ value.

To evaluate the trigger efficiency, events selected by a dedicated L1 trigger are used. Events featuring a $Z^0 \to \mu^+\mu^-$ decay are further selected. Muons are invisible to the trigger algorithms in question, since they deposit negligible energy in the calorimeters. Therefore, the momentum of the muon-muon system $p_{\mathrm{T}}(\mu\mu)$ provides a handle for the missing transverse energy seen by the trigger algorithms. Figure 3.9 shows the efficiency of these trigger chains as a function of $p_{\mathrm{T}}(\mu\mu)$. All chains start at zero efficiency for low $p_{\mathrm{T}}(\mu\mu)$, as the missing

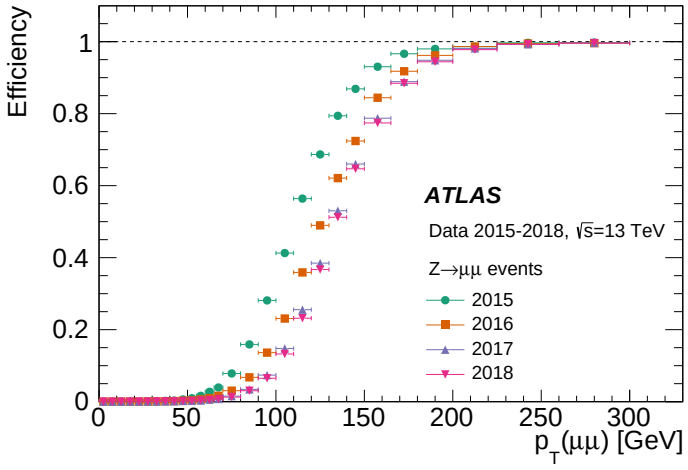


Figure 3.9.: Trigger efficiency as a function of p_{T} of the $\mu\mu$ system from Z^0 decay. Shown are the respective lowest unprescaled trigger chains. Figure taken from [119].

transverse energy is below the configured thresholds. As $p_{\mathrm{T}}(\mu\mu)$ approaches the thresholds, the chains rise in efficiency, roughly between 100 GeV and about 200 GeV, above which they are approximately fully efficient.

To adjust the acceptance rate of these triggers, their configuration and thresholds were adjusted between the years. The 2015 trigger featured an HLT threshold of 70 GeV, which was tuned to 90 GeV and later 110 GeV for 2016. In 2017, the lowest unprescaled chain gained the pile-up suppression discussed previously, allowing for thresholds between 90 GeV-110 GeV in that year. The corresponding threshold in 2018 remained at 110 GeV. The varying thresholds of the chains explain the shift in turn-on observed in figure 3.9.

3.4.3. Data acquisition system

The purpose of the DAQ system [120] is to orchestrate the transfer of data from the detector to its eventual destination. It is part of the online computing environment deployed by ATLAS (see section 3.3.1). As of Run 2, the previously mentioned HLT is highly integrated in the DAQ chain. Frontend electronics buffer the detector signals until an L1 decision is available. In case of an accept, the readout system transfers the signals from the frontend into dedicated buffers. The HLT requests information from these buffers as needed. When an event is accepted at HLT, the buffers in the readout systems are cleared, or the event data is transferred for permanent storage and offline processing at CERN's Tier-0 [108] center.

At an HLT target output rate of 1.2 kHz and an event size of 1 MB for the associated streams, this results in a data rate of 1.2 GB s^{-1} that needs to be handled by the DAQ. It is able to buffer 24 hours worth of event data, before transfer to Tier-0. This allows the HLT output rate to peak over 2.5 GB s^{-1} at the start of an LHC fill, before dropping off. The average output rate is low enough to fit into the DAQ buffer. During Run 2, about 19×10^9 events were recorded in the main physics stream.

3.5. Event reconstruction

Event reconstruction describes the process of analyzing the raw recorded data and extracting information about the physical processes that occurred in any given event. It is generally a computing resource intensive task, which is implemented using a plethora of algorithms. Some of them are tailored and specialized toward a specific type of particle or group of physical processes. The output of the reconstruction is a version of the event data, in which raw electronic signals have been converted into a representation of the physical objects found in the event. A rough separation in two stages can be made. The first stage produces low-level physical signatures like tracks or calorimeter clusters. A second stage uses these low-level objects to construct higher-level objects corresponding to particles.

This section gives a brief overview of the reconstruction chain starting with the lower-level in section 3.5.1, followed by the higher-level in section 3.5.2. The focus is on the aspects relevant for the analysis found in part III of this thesis. Additionally, since the topic of part II is track reconstruction entirely, a detailed description of it is located in chapter 4.

3.5.1. Reconstruction of detector signatures

At the first stage of the reconstruction, raw data that is recorded from the detector is converted into objects with a physical interpretation. In most cases, this raw data is in the form of digitized electronic signals. Already at this stage, knowledge about the physical phenomena in play is used to make assumptions about how to treat the data. The ATLAS reconstruction software is embedded in the Athena software environment.

Tracks

The reconstruction of particle tracks uses the measurements from the dedicated tracking detectors that are part of ATLAS. Here, the digital signals from the Pixel detector, SCT and TRT are first converted into clusters, in the case of the silicon detectors, and drift circles, for the TRT. These measurements are then converted into three-dimensional points in space. From the collection of space points, a seed finding algorithm constructs triplets that are compatible with originating from the interaction region, and have a minimum momentum. An exploratory algorithm iteratively adds more measurements to these triplets to form track candidates. These track candidates are then ranked and filtered, before a final track fit extracts a precise parameter estimate. Following this, an attempt is made to extend tracks into the TRT to include additional information measured there. A more detailed description of the track reconstruction chain is the topic of chapter 4.

Calorimeter clusters

A central part of many calorimeter based reconstruction algorithms are *topological clusters* [121, 122], as a means of suppressing both electronic noise from the calorimeter readout, as well as effects caused by pile-up. Seed cells are identified based on a high signal-to-noise

ratio. These seed cells are then expanded by recursively collecting neighboring cells in three-dimensions, that meet another signal-to-noise threshold. Specialized calibration is applied to obtain the best possible estimate of the cluster energy. Depending on the use-case, clusters are either calibrated to the electromagnetic scale, or a specialized calibration correcting for differences between the hadronic and electromagnetic calorimeters is used.

3.5.2. Reconstruction of physics signatures

The second stage of reconstruction uses lower-level signatures, including tracks and calorimeter clusters, in order to build a representation of the physical particles that are likely to have caused them. In ATLAS, object reconstruction can loosely be divided into four domains:

1. Electron / Positron and photon reconstruction
2. Jets and missing transverse energy reconstruction
3. (Anti-)Muon reconstruction
4. (Anti-)Tau reconstruction

Here, the outputs of some domains are used as inputs in other domains. Apart from a measurement of charges of particles, the reconstruction does not differentiate between particles and anti-particles, and neither does the following description. Also, this section focusses on the reconstruction of physics objects that are relevant to the analysis presented in part III: electrons, muons and jets. In all three cases, the reconstructed objects are subjected to a calibration procedure, that ensures the measured energies reproduce known resonances.

Electrons and photons

Electron reconstruction in ATLAS [123] uses both calorimeter and track information. Since electrons produce tracks as well as calorimeter signals, a calorimeter cluster with a matching track is considered as an electron. Special dynamically sized *superclusters* are formed based on topological clusters that are matched to tracks. These dynamic clusters are more suited to capture an electron's energy including potential bremsstrahlung. Depending on the η range, matched tracks are refitted using a dedicated approach more suited for tracks undergoing bremsstrahlung (see section 4.7). At the same time, potential photon conversion vertices are reconstructed. The final four-momenta are composed of the calorimetric energy measurement, and directional information from the tracker, if the signature is associated with a track.

In the simplest case, a photon is measured as an energy deposit in the electromagnetic calorimeter, without an associated track. While photons do not leave signals in the tracking sensors, they can still undergo interactions when passing through the passive material of the ID. Photons can produce electron-positron pairs in the vicinity of nuclei. If this occurs while still inside the sensitive ID volume, the electron-positron pair will produce track signals. In contrast to prompt particles, these signals will point to the interaction region, but only appear

in the location where the pair-production occurred. As a consequence, calorimeter signals with tracks that can be associated with a likely photon conversion vertex are reconstructed as a *converted* photon.

Lower amounts of material in the detector are beneficial for electron and photon reconstruction. Both bremsstrahlung as well as pair-production require the presence of nuclei, and lower reconstruction performance. Therefore, limiting passive material can improve reconstruction of these particles.

Muons

Muon reconstruction [124] in ATLAS is executed independently in the ID and MS, as muons produce signatures in both of them. In the ID, muons are reconstructed using the regular algorithms without any specific modification. In the MS, reconstruction begins by searching for hit patterns in the MDTs using a Hough transform. From a straight-line fit, an MDT *segment* is constructed. In the CSCs, segments are built using another combinatorial search method, requiring segments to be compatible with the interaction region.

These segments and hits in the RPCs and TGCs are fitted together, seeded by segments in the middle layers. Subsequently, the muon candidate track is extended outwards. Track candidates can share segment and hits, but are filtered based on these and other quality criteria. If a candidate contains segments from three different layers, it is kept in any case to ensure high efficiency. A final global χ^2 fit is performed and candidates are again filtered based on the resulting χ^2 value. Refits are attempted while removing hits contributing strongly to the χ^2 value, and also adding additional compatible hits.

Following the MS muon reconstruction, a set of four combination methods exists, using different detector components. One of these, the *combined muon* algorithm uses ID muon tracks. Here, a combined global track refit is performed using the hits from both the ID and MS. Again, additional MS hits may be added when found compatible and improving fit quality. Overlaps between this type of muon, and muons reconstructed using the other three approaches, are removed.

Jets

As mentioned in section 2.1.4, jets are the result of partons showering, and finally forming hadrons in a process called hadronization. Unstable hadrons produced in this manner will also undergo decay into stable particles. The ensemble of particles originating from a parton in this way is called *jet* As this is a very common process in proton-proton colliders like the LHC, reconstruction of these jets is crucial. The jet constituent particles are usually collimated bundles, making clustering techniques helpful for their reconstruction.

In ATLAS, jet reconstruction [122] primarily uses topological clusters, that were mentioned before. Two main reconstruction approaches are used in ATLAS. The first one, the anti-k_t clustering algorithm [125], uses a radius parameter to quantify distance between the calorimeter clusters. Another input is a distance measure between a cluster and the beam line. The

algorithm works by iteratively finding the pair of input objects that have the lowest distance, and combining them. Whenever a, possibly combined, object is closest to the beam line, it is labeled as a jet and added to the algorithm output as a jet. The resulting jet candidates can be assigned effective momentum and direction values based on their constituents. Tracks are only combined with these types of jets after they have been clustered. A radius parameter of $R = 0.4$ is most commonly used in ATLAS. This first approach was the default one in ATLAS until very recently. The second approach uses a *particle-flow* [126] technique. It explicitly uses track information for pile-up suppression, and is now recommended for analyses.

3.5.3. Particle identification and isolation

Aside from the mere reconstruction of the physics objects themselves, a crucial input for many types of analysis is the degree with which a detector signature is believed to be originating from a given particle. The reconstruction algorithms in the previous section combine signals from different detector components and assume any eligible combination as valid candidates.

During and after the reconstruction stage, the validity of this assumption is quantified using a set of *identification* procedures that are executed on the particle candidates. In most cases, each procedure has a number of working points, which target different efficiencies or purities for the associated objects.

Electron identification

Electron identification [123] uses a likelihood-based approach. During electron reconstruction, calorimeter clusters that can be associated with a charged particle track are treated as electron candidates. Prompt electrons need to be separated from objects like hadronic jets and converted photons, and actual electrons produced in hadron decays, as these produce signatures very similar to that of prompt electrons.

Variables from the track associated to the electron, lateral and longitudinal shower shape variables and spatial compatibility between the tracker and calorimeter signatures enter the procedure. All the variables are combined into a likelihood discriminant comparing hypotheses using a likelihood ratio. A number of working points exist that classify the electrons as *signal* or *background*, and are roughly separated into three categories: *loose*, *medium* and *tight*. In addition to the likelihood itself, fixed cuts are also incorporated. The working points are generally designed to meet specific efficiency levels (93 %, 88 % and 80 %). Tighter working points are defined such that the populations they define are strict subsets of the looser working points[12]. Additional variations of these working points exist, which can include more specific requirements.

Muon identification

Muon identification [127] consists of a set of quality requirements, that are meant to provide background rejection. Major background sources include muons originating from pion and

[12] E.g. all *tight* electrons are also *loose* electrons.

kaon decays. On the other hand, the goal is to achieve high efficiencies for muons originating directly from the primary interaction.

As the aforementioned processes produce tracks that do not follow ideal curved trajectories, such muons should yield a worse fit quality in the combined ID-MS fit. Therefore, these parameters are used for the identification. As for electrons, a set of working points *Loose*, *Medium* and *Tight* are defined. They are defined so that the muons selected by tighter working points are subsets of the looser working points. The *Medium* working point is the default for most analyses, which minimizes systematic uncertainties involved. *Loose* is designed to maximize efficiency, while *Tight* is designed to maximize purity. Additional, more specialized working points exist.

Particle isolation

Particle isolation is often an additional quantity relevant for analyses. In many cases, analyses select objects that are produced individually, rather than for example being part of a jet or other signatures. To this end, an isolation variable is frequently defined. In the simplest form, it is the sum of all transverse momenta of particles within a specific ΔR cone around the momentum vector of the primary particle. Thresholds are then set on the fraction of the total sum of p_{T} in the cone relative to the particle p_{T}. If the particle dominates the sum of momenta, it can be considered to be isolated. One extension of this criterion is the definition of a dynamic cone size depending on the particle momentum. In principle, the concept of isolation is generically applicable to all particle types. Nevertheless, specially tuned isolation criteria exist for different particle types, more closely matching the respective particle characteristics.

3.5.4. Missing transverse momentum

Certain particle types are invisible to the detector. An example of this are neutrinos, which do not interact with the detector elements, and exit the experiment without being detected at all. To measure these particles, the fact that momentum should be balanced in the transverse plane, due to momentum conservation, is exploited. By taking the vector sum of all visible particles, and negating it, the vector sum of invisible particle momenta can be approximated. The $E_{\mathrm{T}}^{\mathrm{miss}}$ algorithm is centrally implemented [128], and calculates

$$E_{x(y)}^{\mathrm{miss}} = E_{x(y)}^{\mathrm{miss},e} + E_{x(y)}^{\mathrm{miss},\gamma} + E_{x(y)}^{\mathrm{miss},\tau} + E_{x(y)}^{\mathrm{miss,jets}} + E_{x(y)}^{\mathrm{miss},\mu} + E_{x(y)}^{\mathrm{miss,soft}} \tag{3.10}$$

in the x and y direction independently. The terms include momenta from electrons, photons, tau leptons, jets and muons. Finally, an additional *soft* term exists, that attempts to pick up any remaining visible momentum not associated with any of the above. Overlaps between the input objects are removed, to prevent counting any momentum contribution more than once.

From these individual coordinates, the total missing transverse momentum can be calculated as

$$E_{\mathrm{T}}^{\mathrm{miss}} = \sqrt{\left(E_x^{\mathrm{miss}}\right)^2 + \left(E_y^{\mathrm{miss}}\right)^2}, \tag{3.11}$$

while the direction in the transverse plane can be expressed with

$$\phi^{\text{miss}} = \arctan\left(E_y^{\text{miss}}, E_x^{\text{miss}}\right). \tag{3.12}$$

3.6. Simulation

In many applications, an expectation of how the detector reacts to certain physical events and processes is necessary. While other techniques exist, very frequently this is achieved by simulating physics processes and evaluating what response this would produce in the detector. This procedure can be divided into two parts. The first one is simulating the actual hard-scatter particle interactions that occur in the collisions. Its outputs are all the particles that would be produced by the reaction, and which would enter the detector volume. The second part is simulating the way the detector would respond to this set of particles passing through it. Both steps use Monte Carlo (MC) techniques to achieve this.

3.6.1. Simulation of processes

The simulation of physical processes is carried out by approximating the probabilistic nature of quantum physics at the fundamental level. As such, all quantities come with associated probability densities, which are modelled in order to simulate a physically accurate outcome.

For particle interactions, this is mostly done using the analytical cross-sections that can be determined from theory. The cross-section is a measure of the probability for a transition from a specific initial state to a specific final state, where both of these consist of a set of particles. In quantum field theory, these transition probabilities are the sum of an infinite series of transition modes (see section 2.1.3). The probability of each one is suppressed by the number of the coupling vertices that are part of the transition. To keep computation feasible, simulations are terminated at a specific order, which introduces uncertainties. Correction factors are determined to map lower orders to higher order precision in some cases.

In proton-proton collisions, simulation is complicated by the fact that the initial state consists of composite, rather than elementary, particles. As a consequence, the simulation needs to model not only the hard-scatter process, but also parton showering, hadronization, and particle radiation at various stages (see section 2.1.4, especially figure 2.3). Only the hard-scatter cross-section can be calculated fully from theory. Parameters related to the factorization of QCD confinement effects need to be chosen and applied. Another aspect is the choice of a PDF to correctly model the proton structure for the simulated interaction. Simulation of parton showers recursively models QCD radiation, until the shower particles drop below a certain energy threshold. Phenomenological models are used to model the conversion of parton shower particles into hadrons, and decays of these hadrons need to be simulated accordingly. Initial and final state radiation also needs to be modelled.

A wide variety of event generators exist, that each have unique strengths and weaknesses. For ATLAS simulation, different generators are often combined to simulate different aspects

of the proton-proton interaction. Examples of event generators include `MadGraph5` [129], `Pythia8` [130] and `EvtGen` [131], which are used in the analysis found in part III of this thesis. Uncertainties in simulation a largely connected to the cross-sections involved, as well as the PDFs. Aside from this, if a correction factor between coupling constant orders is used, the uncertainties arising from its determination also need to be considered.

3.6.2. Simulation of detector response

Given the simulation of particle interactions at the fundamental level, the response of the detector to the produced particles needs to be simulated. This procedure also uses MC techniques to sample relevant probability distributions. In many cases, and in ATLAS in particular, the `Geant4` [132] software is used. The simulation of the detector performance with `Geant4` is resource intensive, accounting about 40 % of the overall CPU resource consumption of ATLAS.

`Geant4` propagates particle trajectories using a numerical integration approach. At each step, the probabilities of all relevant effects are evaluated, incorporating the surrounding material at the step's location. If a process is decided to take place, the particle is modified accordingly. The propagation then continues. It also handles secondary particles, that can be produced as part of the interactions.

When simulated particles pass through active elements, the amount of energy that they deposit is recorded. Using digitization algorithms, these simulated energy deposits are converted into detector readout signals, that closely resemble the actual signals read out during data taking. This simulated readout data is then handed over to the reconstruction algorithm, exactly like real data.

To enable the retrieval of properties associated with the true particles that were simulated, their information is stored alongside the simulated readout data and reconstruction output. In this way, it is possible, for example, to determine whether a reconstructed particle belongs to an actual simulated particle.

3.7. Calibration and corrections

When using simulated data, additional corrections are carried out to ensure a precise match between simulated and real quantities. One such correction is the energy calibration. This is achieved by comparing spectra in simulation and data, with the goal of reproducing known quantities such as resonance energies. Again, particles and anti-particles are not distinguished.

3.7.1. Calibration

For electrons and photons [123], $Z^0 \to e^+e^-$ decays are used to determine two energy related corrections. The first is the energy scale. Here, the intention is to recover the correct peak energy, determined experimentally. By optimizing the agreement of the invariant mass distribution between simulation and data, the energy scale and resolution calibration can

be determined. On one hand, data is corrected such that its energy scale aligns with the simulated one. On the other hand, the simulation should correctly model the resolution found in data, and therefore the simulation is corrected. Muon calibration [124] is conducted in a similar fashion as for electrons and photons, using $Z^0 \to \mu^+\mu^-$ and $J/\psi \to \mu^+\mu^-$ events.

Jet calibration [133] involves suppression of pile-up contributions, in addition to correcting the energy scale and resolution. Using vertex information, jet axes are recalculated to point at the hard-scatter vertex, rather than at the interaction point. After quantifying the pile-up energy by means of analyzing the overall energy density, an estimate of the pile-up contribution is subtracted from the four-momentum of each jet. Subsequently, an energy calibration is applied that corrects for parts of the hadronic shower not seen by the calorimeters. The calibration is based on simulated estimates of the invisible energy fraction. *In-situ* techniques using the balancing of momenta in the transverse plane exist to account for additional effects.

3.7.2. Efficiency corrections

Differences between data and MC simulation also occur when considering any kind of selection procedure. Here, as the selection can depend on details of the properties of the event or particle, any mismatch between data and simulation can lead to differing efficiencies. One example is the efficiency of the trigger selection. In MC, this selection has to be emulated, as the actual trigger does not execute in this context. Other examples include the identification and isolation of particles.

At particle level, efficiency correction determination often involves *tag-and-probe* methods. Here, a pair of detected objects from a known decay resonance is selected. One of the objects, called *tag*, is required to be identified with a high degree of certainty. By restricting object pairs to have an invariant mass consistent with the decay resonance, the tight criteria placed on the tag object can be implicitly demanded for the other object, the *probe*. However, no explicit criteria aside from the pairing are applied to the probe. The probe can then be used to measure the efficiency of any object level selection criteria, as no explicit selection has been applied to it. By comparing the efficiencies between data and simulation, the correction factors can finally be derived.

3.7.3. Pile-up effects

Care has to be taken to correctly model the influence of pile-up on simulated events. Pile-up interactions are added to the simulated events, to model the way the reconstruction handles them. However, an assumption on the amount of pile-up interactions has to be made. The quantity of interest in this respect is the mean number of interactions per bunch crossing, $\langle\mu\rangle$. This quantity varies, both during the overall data-taking campaign, but also within a single data-taking run as beam luminosity drops, or beam parameters change.

Ideally, the pile-up $\langle\mu\rangle$ distribution of the simulated samples matches that of the data. In practice, however, simulation is needed before the precise pile-up profile is known. To compensate, simulation is run with a fixed distribution of $\langle\mu\rangle$, and then a reweighting procedure

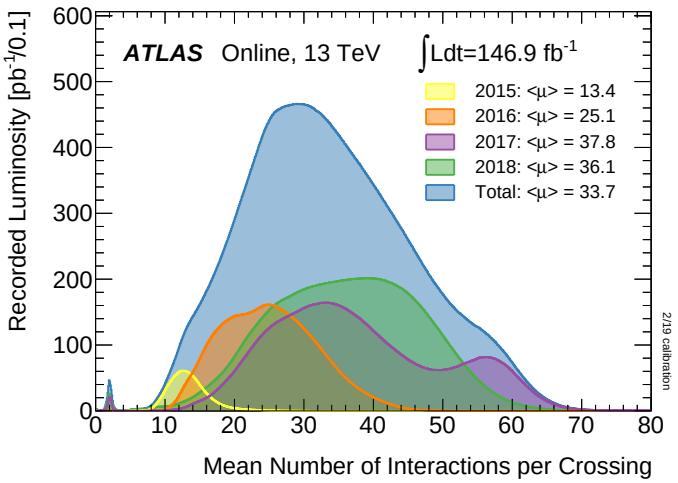


Figure 3.10.: Plot of the distributions of $\langle\mu\rangle$ across the years of the Run 2 data-taking campaign. The effective combined distribution is also shown. Figure taken from [134].

is used to model the distribution eventually observed in data. As an optimization step, after the actual data profile is known, simulation can be rerun with a more closely matching distribution to begin with, reducing the impact the weighting procedure has on the uncertainties. Figure 3.10 shows the pile-up distributions for 2015 through 2018. A peak at $\langle\mu\rangle \sim 30$ can be observed in the combined distribution and a maximum of $\langle\mu\rangle \sim 70$ can also be seen.

3.8. Luminosity measurement

As previously indicated, one critical ingredient in the analysis of recorded data, especially when using simulation, is the integrated luminosity. This quantity is a measure of the amount of data that was recorded during operation. When using simulated processes, it is needed to correctly normalize the expectations derived from MC simulation. Using the cross-section σ of a simulated process, the expected number of events N is

$$N = \sigma \cdot \mathcal{L}_{\text{int}} \tag{3.13}$$

with the integrated luminosity $\mathcal{L}_{\text{int}}$.

Several ways exist to measure this quantity, as described in [135]. In general, they are based on measuring the overall event activity. The bunch luminosity can be obtained using the relationship

$$\mathcal{L}_b = \frac{\mu_{\text{vis}} f}{\sigma_{\text{vis}}}, \tag{3.14}$$

with the visible interaction rate μ_{vis} per bunch crossing, the visible cross-section σ_{vis} and the revolution frequency f. While the latter two can be calculated, the former is measured during operation.

The measurement of μ_{vis} occurs in three ways in ATLAS. The BCM[13] contains diamond sensors very close to the beam line. Using a very high readout frequency, the event activity can be recovered from the sensor activation. A dedicated luminosity monitor, LUCID[14] [136], is located further away from the interaction point, but again very close to the beam line. Here, Cherenkov radiation is used to measure the amount of passing charged particles. Finally, the tracking system can be used in combination with a random trigger. This also results in a measurement of the overall amount of particles seen in the detector.

All methods require a calibration to map event activity to μ_{vis} and then the luminosity. In a so-called van-der-Meer (vdM) scan, the proton beams are separated in a controlled way. Beam intensity is lowered to reduce pile-up. The calibration can be recovered from the observed event activity at a given beam separation distance.

The nominal luminosity measurement is carried out continuously during data taking using LUCID, and is integrated over Luminosity Blocks (LBs), during which the conditions are assumed to be constant. These LBs can be as short as 60 s. The total integrated luminosity after data quality requirements amounts to 58.5 fb^{-1} in 2018, with an uncertainty of $\pm 2.0\,\%$ [137]. In Run 2, from 2015-2018, 139.0 fb^{-1} $\pm 1.7\,\%$ of integrated luminosity were recorded.

[13] Beam Conditions Monitor
[14] LUminosity measurement using a Cherenkov Integrating Detector

4. Reconstruction of charged particle trajectories

Track reconstruction is the process to recover the properties of a charged particle from a set of measurements caused by interaction with some form of sensitive detector. The goal is to find which measurements are likely to have been caused by which particle, group them accordingly, and estimate the associated trajectory. Such charged particle trajectories form the basic input to the majority of higher-level reconstruction procedures in many cases.

Chapter 4 introduces general concepts for this purpose. While most of the aspects are reasonably generic, the description is given from the perspective of track reconstruction at collider experiments. In particular, the stages described closely follow the chains deployed in the tracking software [138] of the ATLAS experiment at the LHC, and are geared primarily toward measurements of proton-proton collision products. CMS uses largely similar detection hardware, and the used reconstruction algorithms also show commonalities when compared to ATLAS, with a few exceptions.

Other LHC experiments such as LHCb and ALICE, and experiments at other colliders, feature different detection hardware. Therefore, their reconstruction chains differ to some extent, although the general strategy remains largely similar.

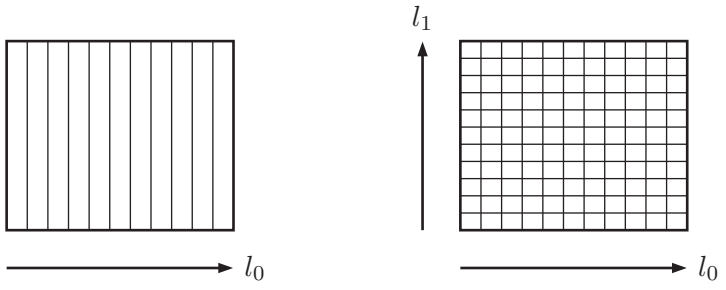
4.1. Charged particle detection

The first step in the chain to reconstruct charged particles is their detection using sensitive elements. Charged particle detection can be achieved in a variety of ways with very different technologies. It can be achieved by measuring the interaction of charged particles with matter. When this occurs, the interacting particle typically ionizes the surrounding material. Particle detectors make use of this fact by converting the resulting charge into a measurable signal in various ways.

Historically, track detection was commonly performed using photographic techniques. Using bubble or cloud chambers, particle tracks can be recorded by photographing the optically visible streaks they cause in the material through windows in the device. Another approach is the use of nuclear emulsion plates that are sensitive to passing particles. In these cases, the reconstruction itself was typically done by humans tracing the streaks on images, and measuring their lengths and curvatures by hand. It is easily comprehensible that this only works with a sufficiently low number of events. In case the carrier material can not be reused, their lifetime also limits the feasibility of long term usage.

Reconstruction of tracks has since largely been migrated to electronic techniques. One very direct way of deploying these is to take digital photos or scans of the aforementioned photographic plates. Image recognition techniques can then be used to extract information. Signals can also be measured electronically, rather than only electronically processing other types of detectors. An example is the use of wire proportional chambers [139], and numerous variations of their concept. Here, a chamber containing a large number of parallel wires is filled with a carefully chosen gas mixture. An electric potential is applied to the anode wires and the cathode walls. Charged particles traversing the volume ionize the gas. This ionization results in pairs of charge, as positive ions and negative electrons are separated. These charges drift toward the electrodes, the wires among them. As they approach the wires, they can be multiplied, depending on the field configuration. The amplified signal is read out for each of the wires. As the position of each wire is known, and the charge drift radius can be derived from the drift time, the particle location can be recovered using a reference time. One notable variation of this approach is the transition radiation detector, which is used in ATLAS in the form of the TRT. As discussed in section 3.2.2, it consists of of *straw tubes* which are filled with an ionization gas mixture and contain wires. Here, the wire is used as the anode, while the tube itself is used as the cathode.

A very common electronic detection approach is the use of semiconducting particle detectors, often made of silicon. When a charged particle traverses such a sensor, it ionizes the material in the depletion zone, caused by the interface of two different semiconducting materials. The result are pairs of opposite charges. These charge pairs are separated by an electric field and drift toward the electrodes. At this point, an electric signal is created which can be amplified and read out. By means of segmentation, the measured signal can be associated with a location on the sensor. Silicon sensors are usually segmented in one dimension (*strips*) or in two dimensions (*pixels*), as can be seen in figure 4.1b.



(a) strip (b) pixel

Figure 4.1.: Illustration of one-dimensional (a) and two-dimensional segmentation (b) of a silicon sensor.

Other technologies include Time Projection Chambers (TPCs), where a large gas volume receives ionization charges from charged particles, which then drift toward electrodes placed at the ends of the detection volume. Segmentation of the electrodes allows localization of the charges that reach them. By measuring the time at which the charges reach the electrodes,

the three-dimensional direction of the trajectory can be measured.

A complicating factor for all technologies mentioned above is pile-up. As the number of particle tracks that are seen by the detector simultaneously increases, their reconstruction becomes more difficult. If dead times are too high, or readouts are too slow, it might not be possible to reliably differentiate between separate particles. Mitigation techniques exist, but depend on the concrete detection mechanism being used.

4.2. Track parametrization

In order to be able to express the properties of a particle's trajectory, a choice of parameters has to be made. The parameters need to be able to express all the relevant quantities of interest. In the presence of a magnetic field, which affects charged trajectories, the global position and momentum, as well as the charge are needed to fully specify the particle properties. In addition, a time parameter can be included, but is not used in this chapter. Apart from the global reference frame, track quantities often need to be represented with respect to a surface. This can be achieved with a parametrization [140] like

$$\vec{x} = (l_0, l_1, \phi, \theta, q/p)^T \,, \tag{4.1}$$

although other parameter conventions exist as well. Figure 4.2 illustrates this choice of parameters. l_0, l_1 are the local coordinates of the corresponding surface, $\phi \in [-\pi, \pi)$ and $\theta \in [0, \pi]$ are the angles in the transverse and longitudinal direction of the global frame, expressed with respect to the current location along the trajectory, as indicated in figure 4.2b. θ is the polar angle relative to the positive z-axis, and ϕ is the azimuth angle in the transverse plane. Finally, q/p combines the charge of the particle with the inverse momentum. In figure 4.2a, the global momentum vector $\vec{p}$ is shown, which can be recovered from the parameters $\vec{x}$ using ϕ, θ and q/p.

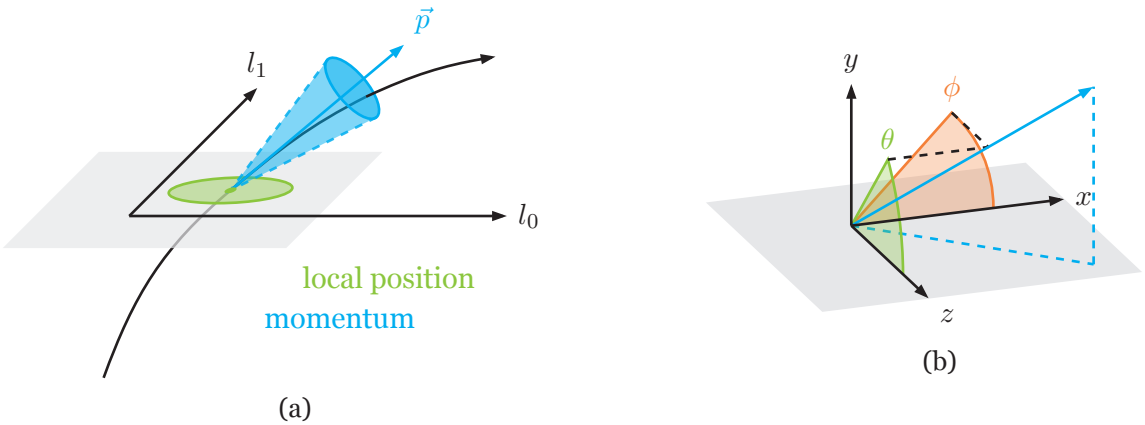


Figure 4.2.: Illustration of the parametrization of a particle track with respect to a two-dimensional surface. (a) shows the local position, global momentum and their corresponding uncertainties. (b) displays the angles ϕ and θ in the transverse and longitudinal planes.

Aside from the nominal quantities captured in $\vec{x}$, the related uncertainties and correlations need to be taken into account as well. They can be expressed as a 5×5 covariance matrix like

$$C = \begin{bmatrix} \sigma^2(l_0) & \mathrm{cov}(l_0, l_1) & \mathrm{cov}(l_0, \phi) & \mathrm{cov}(l_0, \theta) & \mathrm{cov}(l_0, q/p) \\ . & \sigma^2(l_1) & \mathrm{cov}(l_1, \phi) & \mathrm{cov}(l_1, \theta) & \mathrm{cov}(l_1, q/p) \\ . & . & \sigma^2(\phi) & \mathrm{cov}(\phi, \theta) & \mathrm{cov}(\phi, q/p) \\ . & . & . & \sigma^2(\theta) & \mathrm{cov}(\theta, q/p) \\ . & . & . & . & \sigma^2(q/p) \end{bmatrix}. \tag{4.2}$$

Here, $\mathrm{cov}(X, Y)$ is the covariance of variables X and Y, while $\sigma^2(X)$ are the regular variances. As the covariance matrix C is symmetric, only the upper right half is shown in equation (4.2). The uncertainties associated with the local position, as well as the momentum direction are indicated in figure 4.2a as an ellipse and a cone around the momentum vector $\vec{p}$, respectively. The latter does not directly correspond to any of the matrix elements shown in equation (4.2), but can be translated. Note that during the conversion from a global to a local parametrization, care has to be taken to correctly account for the uncertainty along the trajectory direction.

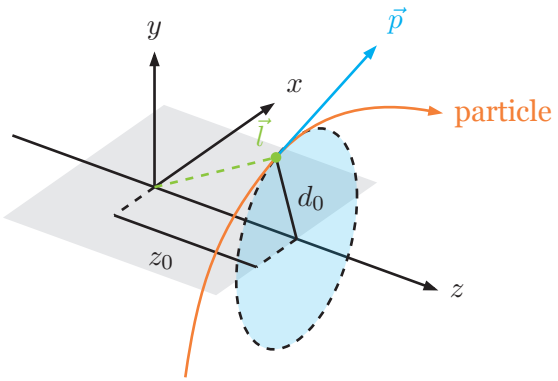


Figure 4.3.: Illustration of the perigee parametrization which uses the point of closest approach relative to a reference point. The impact parameter d_0, the position $\vec{l}$ and the momentum vector $\vec{p}$ are shown.

Different types of surfaces are used throughout the tracking chain, while the most common one is a simple plane surface used to describe the planar silicon sensors. One particular other type is of importance. The *perigee* parametrization, shown in figure 4.3 can be understood as a two-dimensional surface similar to a cylinder surface. Using the point of closest approach of a track as the local position, the impact parameters can be used to describe its location. As seen in figure 4.3, z_0 and d_0 are the impact parameters in the longitudinal direction and transverse plane. d_0 can be thought of as a signed radius, where the sign is determined by the direction of the momentum vector $\vec{p}$ relative to the z-axis like

$$\mathrm{sign}\left[\vec{l} \cdot (\vec{e}_z \times \vec{p})\right], \tag{4.3}$$

with the point of closest approach $\vec{l}$ in the global coordinate system.

4.3. Particle propagation

One of the central pieces required for track reconstruction is the ability to calculate the trajectory of a charged particle, given its properties at a given point. This process, called *particle propagation* or *extrapolation*, is used to predict a particle's properties after it has travelled a certain distance. In many cases, the projected intersection with various types of surfaces is desired. The trajectory of a charged particle is governed by the magnetic field through which it travels, as well as any material effects. In case of a homogeneous magnetic field, and in the absence of material interaction, the particle follows a helical trajectory. Such a helix can be calculated purely analytically, although intersections require numerical methods nevertheless.

Often, and in particular in ATLAS, magnetic fields are not homogeneous, however. The ATLAS solenoid (see section 3.2.2) is capable of producing a homogeneous 2 T magnetic field in its center, but has inhomogeneities at its boundaries. The toroid magnet systems also only produce a homogeneous field in a narrow volume. In the presence of such changing fields, the corresponding differential equations of motions need to be solved using numerical integration techniques.

4.3.1. Numerical integration

In the ATLAS tracking software, numerical integration is done using the Runge-Kutta-Nyström [141–143] method. Commonly used in the variant at fourth order, it describes how to calculate a solution to an initial value problem that can be formulated generically like

$$\frac{dy}{dt} = f(t, y), \qquad y(t_0) = y_0, \tag{4.4}$$

where y_0 refers to the initial value of y at t_0, and $f(t, y)$ is the functional form describing the dynamics. The method then successively approximates the analytical solution $y(t)$ in a stepwise fashion. At each step (t_n, y_n), the goal is effectively to approximate the next step $y(t_{n+1})$. Using a step size h, the algorithm evaluates the function f at four points k_{1-4}:

$$\begin{aligned} k_1 &= f(t_n, y_n) \\ k_2 &= f\left(t_n + \frac{h}{2}, y_n + h\frac{k_1}{2}\right) \\ k_3 &= f\left(t_n + \frac{h}{2}, y_n + h\frac{k_2}{2}\right) \\ k_4 &= f\left(t_n + h, y_n + hk_3\right). \end{aligned} \tag{4.5}$$

Looking at figure 4.4, the meaning of these four points in relation to the step size h can be understood. k_1 is the derivative at the current location, $k_{2,3}$ use k_1 and k_2 respectively to calculate two envelope derivatives at $h/2$ and k_4 uses k_3 to make an estimate of the derivative

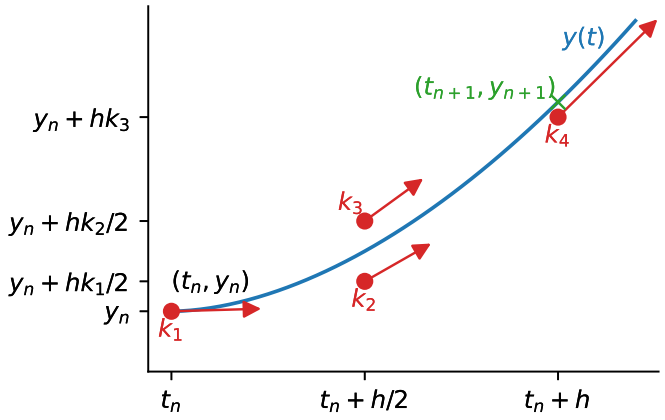


Figure 4.4.: Illustration of the Runge-Kutta-Nyström method approximating a first order differential equation. Shown is the calculation of an estimate y_{n+1} at $t_{n+1} = t_n + h$, based on the current step (t_n, y_n). Shown are the four distinct points at which function $y(t)$ is evaluated, and which are blended to form the estimate.

at h. Combining k_{1-4}, (t_{n+1}, y_{n+1}) can be calculated as the approximation of $y(t_{n+1})$ like

$$\begin{aligned} y_{n+1} &= y_n + \frac{1}{6}h(k_1 + 2k_2 + 2k_2 + k_4) \\ t_{n+1} &= t_n + h \end{aligned} \tag{4.6}$$

by effectively averaging the four derivatives. It is apparent that the step size crucially influences the accuracy of the approximation. A large step size weakens the approximation, especially if the magnetic field changes strongly. On the other hand, a too small step size will negatively affect the execution time of the algorithm.

The Runge-Kutta-Nyström method from above can be adapted [144–146] to handle second order differential equations, as is needed for the equations of motion in question,

$$\frac{d^2\vec{r}}{ds^2} = \frac{q}{p}\left(\frac{d\vec{r}}{ds} \times \vec{B}(\vec{r})\right) = f(s, \vec{r}, \vec{T}), \qquad \vec{T} \equiv \frac{d\vec{r}}{ds}, \tag{4.7}$$

with the global position $\vec{r}$, the path element s, the normalized tangent vector $\vec{T}$ and the magnetic field $\vec{B}(\vec{r})$ at the global position. A slight modification of k_{1-4} is also required, incorporating the first derivative of $f(s, \vec{r}, \vec{r}')$, finally leading to

$$\begin{aligned} \vec{T}_{n+1} &= \vec{T}_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4) \\ \vec{r}_{n+1} &= \vec{r}_n + h\vec{T}_n + \frac{h^2}{6}(k_1 + k_2 + k_3). \end{aligned} \tag{4.8}$$

As described in [146], a strategy exists to dynamically adapt the step size according to the magnetic field strength, with the definition of a target accuracy that the algorithm tries to achieve. Here, the step size h will successively be decreased and the approximation recalculated until the accuracy goal is achieved. Even with these additional calculations, the approach

is still preferable over a consistently low step size.

4.3.2. Covariance transport

Aside from the prediction of the track parameters at a given path length, a key ingredient to many dependent applications are the uncertainties in the form of the associated covariance matrix C introduced in equation (4.2). Conversions between covariance matrices $C^i \to C^f$ can generally be achieved like

$$C^f = J \cdot C^i \cdot J^T, \tag{4.9}$$

using the Jacobian matrix

$$J = \begin{bmatrix} \frac{\partial l_0^f}{\partial l_0^i} & \cdots & \frac{\partial l_0^f}{\partial (q/p)^i} \\ \vdots & \ddots & \vdots \\ \frac{\partial (q/p)^f}{\partial l_0^i} & \cdots & \frac{\partial (q/p)^f}{\partial (q/p)^i} \end{bmatrix}, \tag{4.10}$$

between initial and final parameters $\vec{x}^i$ and $\vec{x}^f$ (equation (4.1)). The task therefore becomes calculating the necessary Jacobians to achieve correct transformation.

One part is the transformation between different coordinate systems, but at the same location along the trajectory. For this purpose, generic Jacobians can be calculated between each coordinate system type, and a common coordinate system. The common coordinate system used for this purpose is the curvilinear frame, which consists of the global direction angles, and a plane surface located at the track location, with its normal aligned with the track momentum. By using Jacobians to the curvilinear frame and the corresponding inverse matrices, conversions between any two coordinate systems can be performed.

The second part is the calculation of the evolution of the covariance matrix during the propagation between surfaces. To this end, a semi-analytical method which calculates the effective derivatives between two consecutive Runge-Kutta-Nyström steps can be used [145]. By accumulating the Jacobian matrices calculated for each step, the effective Jacobian between the starting point and the destination can be obtained.

4.3.3. Material effects

Charged particles interact with matter as they pass through it. Since particle detectors inevitably consist of some form or material, this effect cannot be completely avoided. By building tracking detectors as light as possible, and arranging passive components, such as services and support structures carefully, the material a particle encounters before being measured can be reduced. Charged particles traversing any form of matter undergo elastic and inelastic interactions with the atomic structure of the material, depending on the particle properties.

In elastic interactions, the particle does not lose a significant amount of energy, while its trajectory is affected. Figure 4.5 shows a sketch of the way multiple Coulomb scattering affects the direction of a particle trajectory. In addition, a shift in the transverse plane relative to

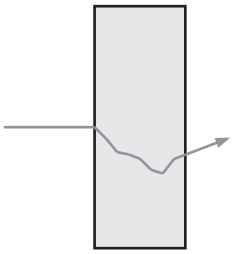


Figure 4.5.: Illustration of the effect of multiple scattering on the trajectory of a charged particle passing through a block of material. Entering from the left, it undergoes a series of scattering events, deflecting the trajectory statistically, before exiting on the right.

the incident direction can occur. As the scattering events occur in statistically independent directions, the means of both the deflection and offset tends toward zero as the number of scatters increases. Therefore, in the numerical particle propagation, this can be accounted for by simply increasing the uncertainties associated with the direction [147], depending on the amount of material encountered.

On the other hand, there are interactions during which the particle loses some of its energy. Relevant processes here are ionization, as well as bremsstrahlung for light particles like electrons. For hadronic particles, hadronic interactions with the nuclei of surrounding material is another process of interest. In such hadronic interactions, the incoming particle often disintegrates, and does not propagate further. Since the size of ionization losses only fluctuates to a small degree for thin layers of material, they can usually be accounted for by reducing the trajectory energy correspondingly. For bremsstrahlung, where fluctuations are much larger, dedicated techniques are needed [148–150].

Two main approaches are implemented in the ATLAS tracking. The first approximates the material interaction by using a description that averages the real material onto thin surfaces across the detector (more on this in section 4.4). When the propagation encounters such a surface, it retrieves the material properties, and executes parametrized modifications to the particle properties and uncertainties. The second approach can be found in [151, 152], where material effects are continuously incorporated during propagation, rather than at discrete locations. The latter approach is especially suited for propagation through volumes of dense material, where the discretization of the material distribution will not work as well.

4.4. Geometry and material modelling

In order to support the various applications that make up track reconstruction, a detailed model of the geometry of an experiment is required. At the lowest level, charged particles are measured when they interact with sensitive elements. In many cases, external information is needed to associate the sensitive element with a position and rotation in the laboratory frame. In case of silicon sensors, the intrinsic information captured by the sensor is restricted to the measurement plane. Using a transformation matrix, this local measurement can be turned

into a global one.

Many particle physics experiments heavily use computerized simulation to define and characterize expected measurements (for an example see section 3.6). Very frequently, the `Geant4` [132] software is used, which includes its own geometry description framework. As the precise simulation of particle interactions with the detector is the focus of these applications, the geometry modelling is highly detailed. During simulation, even very small details of the physical hardware can be crucial, and are often included in the geometry description. An example for this are readout chips on silicon sensors, or cooling elements. Figure 4.6a shows a sketch of such a detailed geometry description. Shown as an example is a *layer* of silicon sensors in a barrel configuration, such as the one found in the Pixel and SCT subsystem of ATLAS. The green rectangles represent the actual sensitive surfaces, while other elements include cooling and readout systems, as well as components for structural support.

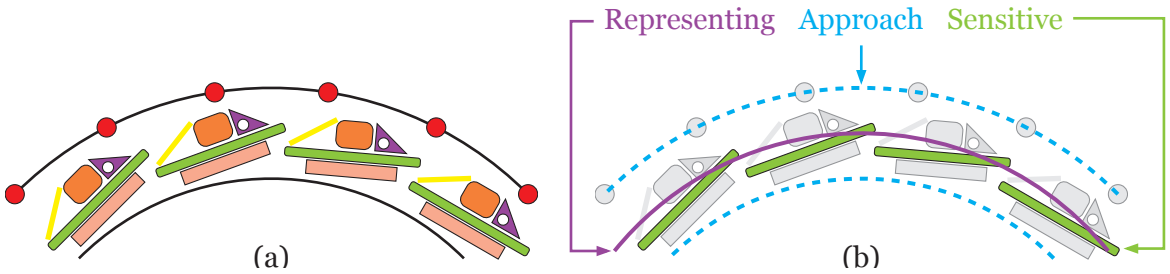


Figure 4.6.: Sketch of the way a fully detailed simulation geometry (a) models passive elements, in addition to the sensitive elements shown in green. (b) shows a simplified version, where all non-sensitive elements are approximated.

In the majority of cases in track reconstruction, this detailed geometry is unnecessary. During track reconstruction, the aforementioned associated information needs to be accessible for measurements, so all sensitive elements need to be included in some form. Passive elements, on the other hand, are only required to factor in material interaction effects (see section 4.3). Moreover, the fully detailed geometry comes at the disadvantage of introducing significant overhead during navigation. In this process, an algorithm attempts to figure out which elements the particle propagation needs to target, as the trajectory is likely to intersect them. With a geometry description this precise, the navigation process becomes a significant performance bottleneck.

As a compromise between modelling accuracy and performance, the ATLAS tracking uses a simplified geometry model [153]. It focusses on the sensitive elements, which are strictly needed, while passive elements are discarded from the explicit description and approximated. Figure 4.6b shows such a simplified geometry. Here, the sensitive elements are still shown in green, and other elements are greyed out, indicating that they are discarded. The sensitive elements are then grouped into layers, as sketched in figure 4.7. How exactly the grouping occurs depends on the concrete geometry, and subsystem. In the barrel region of the ATLAS silicon subsystems, the layers have the shape of cylinder surfaces with increasing radii. This example is shown in the figure in the transverse plane at radii $r_{1,2,3}$. In the endcaps, where

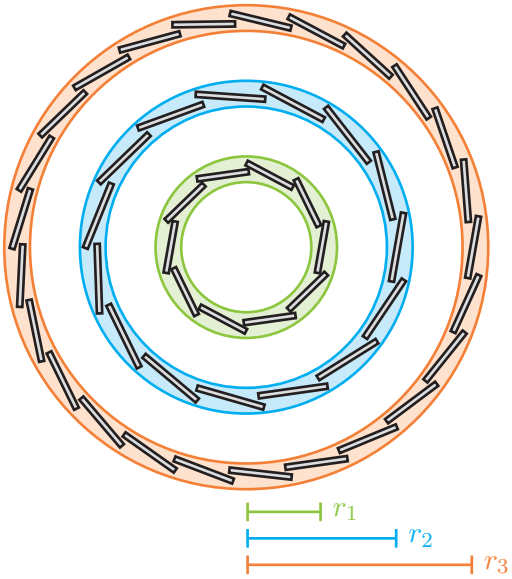


Figure 4.7.: Sketch of the way sensitive elements are grouped into layers. Shown is an xy-view of a number of sensors, arranged as in the ATLAS silicon detector barrels. The grouping is based on their mounting radius. The layers are indicated in different colors.

modules are arranged on disks, these are used as the layer shape. An illustration of endcap disk layers can be found in figure 4.8, where six disks are located at six distinct positions in $\pm z_{1,2,3}$, and shown in different colors.

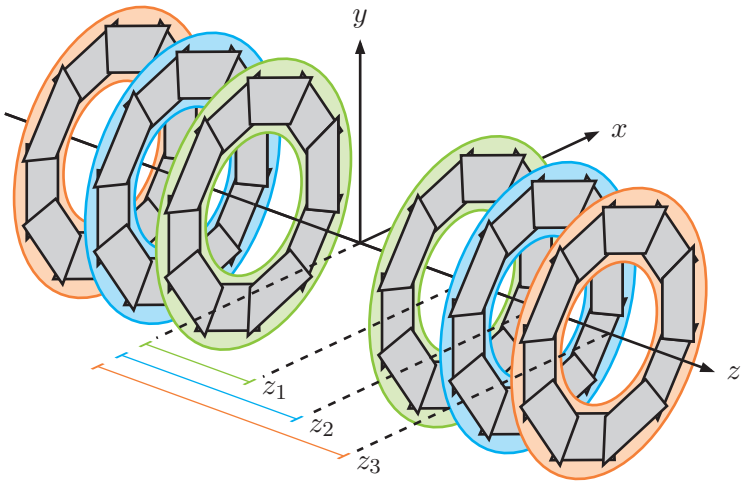
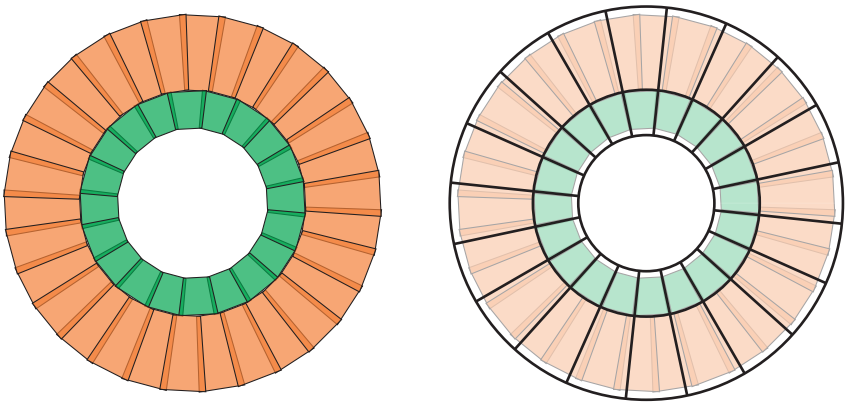


Figure 4.8.: Sketch of the way sensitive elements are grouped into layers. Shown is a view of a number of sensors, arranged as in the ATLAS silicon detector endcaps. They are grouped into disks based on their mounting position in z. The layers are indicated in different colors.

During particle propagation, the navigation makes use of this layer system. Each layer contains a binned structure, which maps a bin to a set of sensitive surfaces that overlap with the bin area. This is illustrated in figure 4.9, where the left picture shows the sensitive surface

structure of an exemplary endcap disk. The picture on the right overlays the binning structure that can be used to enable fast retrieval of compatible sensitive surfaces. By performing a simple bin lookup, the navigation can ascertain which sensors it needs to attempt propagation to.



(a) (b)

Figure 4.9.: Illustration of the binning structure that is used to subdivide layer surfaces. (a) shows two sensor rings of different radii grouped into one disk layer. (b) overlays the binning structure that the navigation queries for compatible surfaces.

Furthermore, layers are grouped into volumes. Each volume loosely corresponds to a region of the detector, such as the Pixel barrel or endcaps in ATLAS. Volumes are set up such that their boundary surfaces always touch another volume. An exception to this is the outermost volume. Each volume's boundary surfaces store which volume is located on their other side, essentially forming portals between the volumes. This glueing enables the geometry navigation between volumes. When the propagation has finished processing a set of layers, it attempts to target the boundary surfaces. Once a boundary surface is reached, the active volume is switched, and the next set of layers is processed.

Care has to be taken to correctly model the passive material, that is initially discarded with non-sensitive elements. For the material effects to be correctly taken into account during particle propagation, the material is projected onto dedicated material surfaces. These material surfaces are spread across the detector geometry. Each layer is created with two *approach surfaces* on either side. Their distance can be interpreted as the thickness of the layer in question. Examples of these approach surfaces can be found in figure 4.6b, at the inner and outer radius. Approach surfaces, and the boundary surfaces between volumes mentioned before, are candidates to receive a projection of the surrounding material. Additional artificial material layers can also be inserted to receive projected material.

The projection procedure works by extrapolating test particles using the fully detailed simulation geometry. During the extrapolation, the material properties of the geometry are sampled in small intervals. Subsequently, the same test particle is extrapolated through the tracking geometry. All material samples are then assigned and projected onto the closest material surface. Finally, the projection is averaged. The exact number and placement of the

material surfaces has to be optimized to yield a sufficiently accurate representation of the inactive material in the detector.

The numerical integration uses these projected material surfaces. Whenever such a surface is encountered in the propagation, the material properties are retrieved, and the corresponding modifications to the trajectory are executed. In case material is supposed to be integrated in a continuous way (as mentioned in section 4.3), volumes can also store an effective volumetric material composition, which is queried by the numerical integration when needed. As the actual physical location of the detection hardware can vary over time, possible misalignment of the sensors needs to be handled correctly, and is discussed in section 4.10.

4.5. Clusterization

The actual track reconstruction procedure itself starts with the conversion of raw inputs that have been read out from the detector. In case of silicon detectors, the readout can either be performed in a binary way, only recording which segments fired, or the amount of charges measured in the segment can be recorded. One method employed in ATLAS is the *time-over-threshold* readout. In all cases, the readout is attached to an identifier uniquely locating the segment on the corresponding sensor.

As a next step, these raw readouts need to be *clustered*, in order to extract an estimate of where particles intersect with the sensor. The general strategy of clustering algorithms follows the Connected Component Analysis (CCA) [154] approach, where subsets of segments are successively grouped into clusters. In case of the Pixel detector, this clustering occurs in two dimensions, corresponding to the segmentation of its sensors. Here, the CCA can either consider all eight surrounding pixels as neighboring a central one, or only consider the four non-diagonal ones, as shown in figure 4.10. The figure only shows the simplest possible cluster starting from the central pixel. In reality, the CCA will iteratively continue from the pixels on the cluster edges. In the SCT, only one dimension is segmented, so the clustering algorithm only operates in this one dimension.

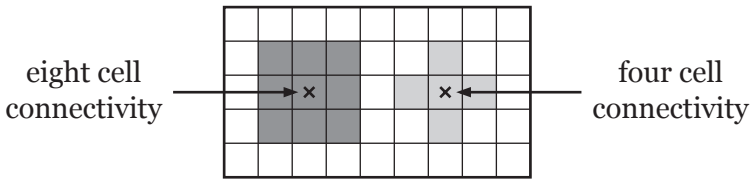


Figure 4.10.: Illustration of both eight and four cell connectivity.

Subsequently, the effective cluster position needs to be estimated. Multiple factors play a role here. First of all, the average position of the cluster can be calculated either using only the geometry position of the segments,

$$\vec{r} = \frac{1}{N} \sum_{i=1}^{N} \vec{l}_i, \tag{4.11}$$

or be weighted by the charge collected in each segment:

$$\vec{r} = \frac{1}{\sum_{i=1}^{N} q_i} \sum_{i=1}^{N} q_i \vec{l}_i. \tag{4.12}$$

Here, $\vec{l}_i$ is the local position of the i-th segment while q_i is its charge. In the case of the ATLAS Pixel detector, estimation of the cluster position was migrated from the charge-weighted approach to a neural-network algorithm [155]. This algorithm can also estimate the cluster uncertainty.

An illustration of the clusterization can be found in figure 4.11, where a pixel sensor is shown to be intersected by a charged particle, entering on the lower left and exiting on the top right. Three cells shown with a red frame receive energy from the particle, but the amount is under the readout threshold. Four other cells receive energy above the threshold and are read out. The clustering will then group these four cells into a cluster, and subsequently estimate the cluster position based on the energy deposited in the cells. Since the SCT is lacking this information, the calculation is purely geometric.

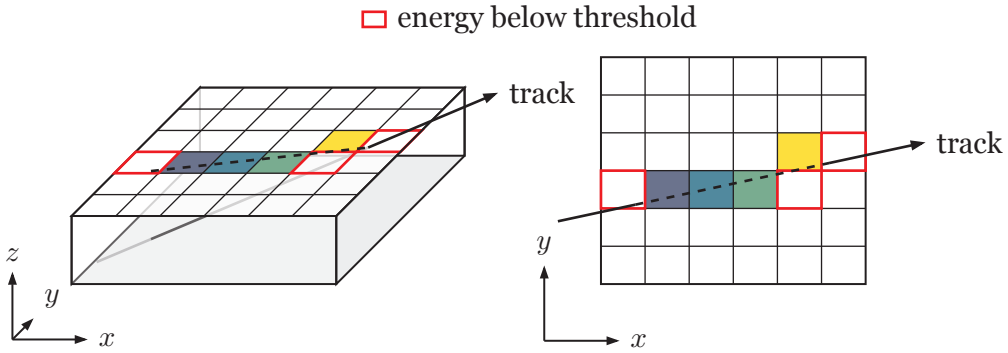


Figure 4.11.: Illustration of the clustering of multiple pixels into a cluster, in a three-dimensional view on the left and a projection onto the xy-plane on the right. A particle enters the center in the lower left, crosses several segments before exiting the sensor on the top right. The cell colors indicate how far along the trajectory they are encountered.

Another factor that needs to be accounted for is the drift direction of the created charges. In addition to the collection field of the sensor itself, the surrounding magnetic field modifies the drift direction by the *Lorentz-angle* θ_{L}. Depending on the field strength, this additional angle can cause segments to be activated that would otherwise not be geometrically within reach of the charges. Other effects, such as the fact that the modules are not perfectly flat, as the geometry description assumes, or cross-talk between readout channels, also play a role at this stage.

In the presence of high event activity, particle intersections on single sensors can be close enough to one another to result in clusters that are not clearly separated from each other. This circumstance can be somewhat mitigated by allowing tracks to share clusters with other particles, which comes at the price of allowing duplicated tracks to some extent. Additionally,

merged clusters typically feature worse position resolution, which manifests itself since it negatively affects the final fit of the track. In ATLAS, the neural-network clustering algorithm determines if clusters are likely to be the result of such a merging. It then attempts to split the clusters, enabling the subsequent steps of the reconstruction to treat them as independent measurements.

4.6. Space point formation and seeding

Seeding is the process of identifying combinations of measurements that are likely to belong to a track. The basic input to this algorithm are space points, which need to be assembled from the raw measurements. To this end, the raw measurements are combined with information provided by the geometry description, such as the location and rotation of the sensors. In this way, the locations, which are restricted to be local to the sensor surfaces intrinsically, can be converted into three dimensional points in space. Figure 4.12 shows an illustration of the information that is consumed for a pixel measurement. Shown are three clusters on a sensor, which are caused by three tracks intersecting it. The corresponding cluster positions are indicated as well, and can be converted to global positions using the inverse of the global-to-local transformation matrix, that is provided by the geometry description.

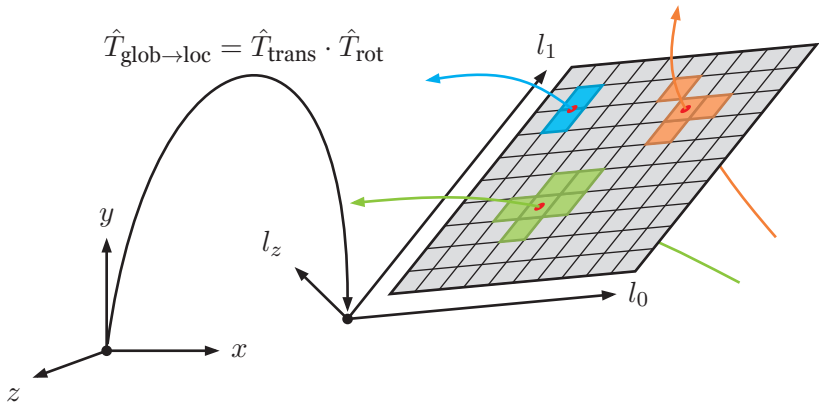
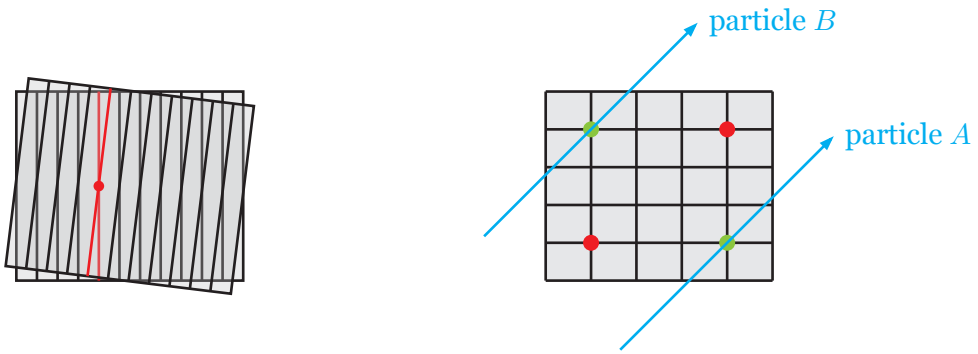


Figure 4.12.: Illustration of a pixel sensor and its local coordinate system in relation to the global laboratory frame. A transformation allows conversion between the two systems. Shown are three tracks intersecting the sensor, alongside clusters that they produce.

In strip detectors like the SCT, on the other hand, only a single dimension is segmented, and an individual measurement is therefore only constrained in one direction on the surface. However, in the SCT, the strip modules are always mounted in pairs, with a stereo angle rotation between the pairs. To form global space points, measurements from both sensors of a pair need to be combined, as is indicated in figure 4.13a. Due to the stereo angle, a two dimensional location on the orthogonal projection plane relative to the two parallel pairs can be found. Using the global transformations of the pair, the combined measurement location can be converted to global coordinates. If multiple measurements are located on a stereo

pair of strip sensors, there exists an ambiguity on how to combine strips to form space points. In figure 4.13b this is illustrated, where two particles leave actual measurements, shown in green. The space point formation algorithm does not know whether to use the correct strip combinations, or the wrong (red) combinations.



(a) (b)

Figure 4.13.: Illustration of how a pair of strip sensors which are rotated relative to one another can be used to recover a measurement's position along the direction without segmentation of the silicon surface. (a) shows the simple case with only one particle. (b) shows an example of ambiguities (red) in the presence of multiple real particle measurements (green).

With a set of global space points converted from the input measurement, attention turns toward figuring out which of them belong to an actual track and how to group them accordingly. This process, often referred to as pattern recognition, can be implemented in various ways. Global methods exist which attempt to cluster space points, such as conformal mapping. In this approach, the space points are transformed into a feature parameter space that reveals patterns for hits belonging to the same track. In the specific example of a Hough transform [156], a parameter space $(\phi, q/p_{\mathrm{T}})$ is used. As a result, each space point is effectively transformed into a line, as a series of combinations of these parameters would lead to the same space point. The lines from a set of space points of a single track will intersect in one common area. Such an intersection can be used to identify which space points originate from the same track. However, this task grows in complexity as detector activity increases and is susceptible to material effects. An approach using the Hough transform is used in ATLAS for identifying hit patterns in the MS during muon reconstruction.

Another group of approaches is the one of seeding and track following. These algorithms differ from the global ones in that they evaluate individual combinations of space points, and successively explore the events. One algorithm from this group is the cellular automaton that iteratively forms chains of space points going from one layer to the next [157], and is used by CMS and ALICE.

In ATLAS [158], an algorithm that operates on coarse subdivisions of the detector is used. This seeding algorithm attempts to find triplets of space points from increasing radii which are likely to belong to the same track. It achieves this by iterating the combinatorial triplets

and successively filtering them. Filtering is performed based on the momentum and impact parameters, which the algorithm attempts to estimate for each triplet.

Under the assumption of a homogeneous magnetic field along the z-axis, charged particles should follow helical trajectories. In the transverse plane, the motion is circular, while it is a straight line in the rz-plane. The transverse impact parameter and momentum can be estimated from the radius of the circle in the transverse plane like

$$d_0 = \sqrt{c_x^2 + c_y^2} - \rho, \tag{4.13}$$

with the circle center (c_x, c_y) and radius ρ. The transverse momentum can be related to available quantities like

$$p_\mathrm{T} \propto \cdot qB\rho \tag{4.14}$$

with the charge q and the magnetic field B. An intersection between the straight line in the rz-plane with the z-axis gives an estimate of the longitudinal impact parameter. In ATLAS, seed triplets are required to fulfill $p_\mathrm{T} > 500\,\mathrm{MeV}$, $|d_0| < 10\,\mathrm{mm}$ and $|z_0| < 250\,\mathrm{mm}$.

An illustration of seeds in the transverse plane is found in figure 4.14. Note that seeds can incorporate hits spread across all of the layers shown, although this can be a configuration parameter.

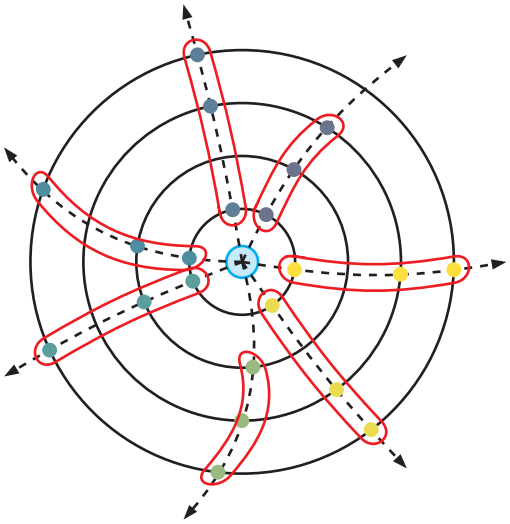


Figure 4.14.: Sketch of seeds in the transverse plane for a number of tracks on four layers. Seeds can combine hits on any three of these layers. The shown seeds appear compatible with having originated in the center of the detector, which is also drawn.

4.7. Track finding and track fitting

In the track seeding and following approach, track candidates are built from the initial seeds. The method employed by ATLAS, the Combinatorial Kalman Filter (CKF), uses the *Kalman*

formalism [159], that is described below. Originally developed for monitoring and steering mechanical systems, it can also be used to iteratively calculate a track estimate [160]. After a set of track candidates has been assembled and filtered (see section 4.8), an additional track fit is usually performed to extract the best estimate of the track. The Kalman formalism can also be used for this, with the addition of a smoothing step that has certain benefits. Other fit strategies exist, such as a global χ^2 fit [161] that minimizes the distances between track-sensor intersections and measurements on all sensors at the same time. One drawback of this method is the necessity to invert very large matrices, which is computationally expensive.

In a track fit, the Kalman formalism can be shown to yield optimal estimates for Gaussian uncertainties. This assumption breaks down when effects like bremsstrahlung come into play. An extension of the Kalman Filter (KF) exists that relies on the individual propagation of a set of trajectories, instead of a single one, to model these biased uncertainties by a sum of Gaussian components. This Gaussian Sum Filter (GSF) [149, 150] achieves better results when fitting particles such as electrons, likely to undergo bremsstrahlung, and is deployed in the ATLAS tracking chain.

Other adaptations, such as filters using a deterministic annealing scheme [148, 162], can alleviate problems with large numbers of hits from noise in a combinatorial fit. By allowing association with many hits at high *temperatures*, and then reducing said *temperature* in a number of iterations, hit assignments can be derived that are robust against premature exclusion of ostensibly incompatible hits.

4.7.1. The Kalman formalism

The basis of the Kalman formalism is a state vector, that can be identified with the set of track parameters $\vec{x}$. Note that the concrete parametrization plays a subordinate role in this context. Rather than building an estimate of the state of a system in real time, a Kalman track fit can be understood as estimating the parameters iteratively in steps. In the track fitting application, each step is defined by a measurement to be included. The evolution of the state vector is described by

$$\vec{x}_k = \mathbf{F}_{k-1}\vec{x}_{k-1} + \vec{w}_{k-1}, \tag{4.15}$$

where the linear function $\mathbf{F}_{k-1}$ transports the state vector at step $k-1$ to step k. $\vec{w}_{k-1}$ is additional so-called process noise that affects the transport additively. Each step has an associated measurement, with the fixed relationship between the measurement and the state vector

$$\vec{m}_k = \mathbf{H}_k\vec{x}_k + \epsilon_k. \tag{4.16}$$

Here, $\mathbf{H}_k$ is the *measurement mapping function*, which transforms the state vector into the measurement space. In the ideal case, this purpose can be achieved by a simple projection matrix, which extracts a subspace of the state vector. Additionally, ϵ_k represents the measurement uncertainty.

The Kalman fit process is divided into different phases:

1. **Prediction** of the state vector at the next step $k+1$ based on the information at the current step k.
2. **Filtering** of the prediction by incorporating the measurement associated to the step
3. **Smoothing** of the state vector by walking back the steps and using information for the subsequent step $k+1$ to improve the estimate at the current step k.

An illustration of these concepts is found in figure 4.15. Here, a series of three sensors is shown with measurements on them. The KF then predicts the track parameters at an intersection, shown in blue. Subsequently, a filtered set of parameters is calculated as a mixture between the measurement and the prediction. Not shown in this picture is the smoothing step.

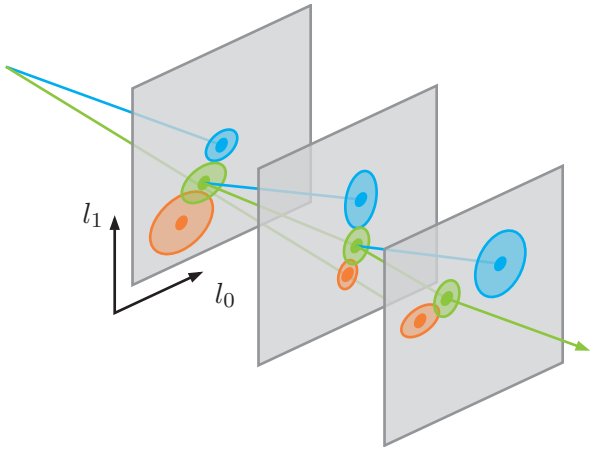


Figure 4.15.: Illustration of the Kalman Filter. Two of the three stages, the prediction and the filtering are shown. The filtering updates the prediction with information from the measurement.

In many cases, the first two phases run in tandem, with prediction and filtering happening alternatingly at each step. The smoothing phase is launched once the last measurement has been encountered. Starting from a state k, first, a prediction of the state vector at the next measurement location is obtained via

$$\vec{x}_k^{k-1} = \mathbf{F}_{k-1}\vec{x}_{k-1}, \tag{4.17}$$

with the linear transport function from above. $\vec{x}_k^{k-1}$ is the prediction of the state vector at step k based on step $k-1$. The next stage is the filtering. Here, the state vector is refined by taking into account the measurement at the current step. Following one of two variants of filtering from [160], the gain matrix formalism, the state vector is updated like

$$\vec{x}_k = \vec{x}_k^{k-1} + \mathbf{K}_k\left(\vec{m}_k - \mathbf{H}_k\vec{x}_k^{k-1}\right), \tag{4.18}$$

with the *Kalman gain matrix*

$$\mathbf{K}_k = \mathbf{C}_k^{k-1}\mathbf{H}_k^{\mathrm{T}}\left(\mathbf{V}_k + \mathbf{H}_k\mathbf{C}_k^{k-1}\mathbf{H}_k^{\mathrm{T}}\right)^{-1}. \tag{4.19}$$

Note that $\vec{x}_k$ is the filtered state vector at step k, based on information from previous steps and step k itself. This is in contrast to $\vec{x}_k^{k-1}$, which is the prediction of the state vector at step k based on $k-1$, and is used to calculate the filtered state vector. One input to these equations is the covariance matrix prediction $\mathbf{C}_k^{k-1}$ at step k based on step $k-1$, which can be written as

$$\mathbf{C}_k^{k-1} = \mathbf{F}_{k-1}\mathbf{C}_{k-1}\mathbf{F}_{k-1}^{\mathrm{T}} + \mathbf{Q}_{k-1} \tag{4.20}$$

in the linear version from [160], with the covariance $\mathbf{C}_{k-1}$ at step $k-1$, and the covariance $\mathbf{Q}_{k-1}$ associated with $\vec{w}_{k-1}$ from above. Also needed is $\mathbf{V}_k$, which is the covariance associated with ϵ_k, effectively representing the measurement uncertainty.

Similar to the state vector itself, the corresponding covariance matrix is also filtered using

$$\mathbf{C}_k = \left(\mathbb{1} - \mathbf{K}_k\mathbf{H}_k\right)\mathbf{C}_k^{k-1}. \tag{4.21}$$

In the smoothing phase, the state vector at step k is improved using the information from the subsequent step $k+1$ using

$$\vec{x}_k^n = \vec{x}_k + \mathbf{A}_k\left(\vec{x}_{k+1}^n - \vec{x}_{k+1}^k\right). \tag{4.22}$$

Here, $\vec{x}_{k+1}^n$ is the smoothed state vector and $\vec{x}_{k+1}^k$ the predicted state vector at the subsequent step $k+1$. Also needed is the *smoother gain matrix*

$$\mathbf{A}_k = \mathbf{C}_k\mathbf{F}_k^{\mathrm{T}}\left(\mathbf{C}_{k+1}^k\right)^{-1}, \tag{4.23}$$

with the predicted covariance at step $k+1$, $\mathbf{C}_k^{k+1}$. Finally, the covariance at the current step k can also be smoothed with

$$\mathbf{C}_k^n = \mathbf{C}_k + \mathbf{A}_k\left(\mathbf{C}_{k+1}^n - \mathbf{C}_{k+1}^k\right)\mathbf{A}_k^{\mathrm{T}}. \tag{4.24}$$

After smoothing, the parameter estimate at the first step contains information from all other measurement states. As mentioned above, in case the uncertainties entering the Kalman fit are Gaussian distributions without biases, the KF can be shown to be the optimal solution minimizing mean square estimation error. However, certain caveats exist. The KF assumes that a linear transport function $\mathbf{F}$ exists that can propagate the state vector. In the presence of inhomogeneous magnetic fields this is not the case. Instead of explicitly applying $\mathbf{F}$ to the state vector for the prediction, the ATLAS KF turns to the numerical integration, discussed in section 4.3. With it, the prediction from equation (4.17) is simply the intersection of the extrapolated trajectory with the next sensitive surface. Aside from this, $\mathbf{F}$ is also used to transport the covariance between steps (see equation (4.20)). Here, the semi-analytical method for covariance transport in the numerical integration can be used. $\mathbf{F}$ can then be

identified with the transport Jacobian accumulated between surfaces.

For smoothing, two possibilities exist to obtain the needed covariances from subsequent measurement steps. Either, the inverse transport Jacobian is used and applied, in a way similar to equation (4.20), or the numerical integration is executed again in an inverse fashion, propagating from the subsequent step to the current one.

4.7.2. Combinatorial Kalman Filter

As mentioned above, the Kalman formalism can be used for track finding. In this case, the smoothing step can be dropped, as the resulting track candidates are likely to be refit regardless, therefore saving some time. The CKF explores the event starting from an initial track seed. It does this by considering not only a single sequence of measurements, but allowing the branching of the fit at each sensitive surface that is encountered. To this end, all or a subset of measurements that are found on each surface are considered. Measurements are selected based on their compatibility with the current state estimate, by using their residuals. A predicted residual

$$\vec{r}_k^{k-1} = \vec{m}_k - \mathbf{H}_k \vec{x}_k^{k-1}, \tag{4.25}$$

and a filtered residual

$$\vec{r}_k = \vec{m}_k - \mathbf{H}_k \vec{x}_k, \tag{4.26}$$

can be defined, depending on which state estimate is compared with the measurement $\vec{m}_k$. Using the filtered residual, an effective χ^2 increment

$$\chi^2_+ = \vec{r}_k^{\mathrm{T}} \left[(\mathbb{1} - \mathbf{H}_k \mathbf{K}_k) \mathbf{V}_k \right]^{-1} \vec{r}_k \tag{4.27}$$

can be calculated. The global χ^2 of the track candidate can be calculated as the sum of all χ^2_+ across the steps. Measurements with a large χ^2_+ are considered as outliers, which have low compatibility with the trajectory. By branching out for measurements below a certain χ^2_+, and following the branches, a tree-like structure of compatible track candidates originating from a track seed is assembled. This feature is shown in figure 4.16, which displays a circular trajectory, and a set of iteratively assembled track candidates. Basic quality criteria can be applied at this stage, to remove bad candidates. A dedicated *ambiguity resolution* step (see section 4.8) selects the candidates most likely to belong to real particle tracks.

4.8. Ambiguity resolution

As discussed in the previous section, track finding describes the process of assembling a set of track candidates from a set of track seeds. Due to the combinatorial nature of track finding, and to achieve high efficiencies, this set of candidates is often large, and contains a non-negligible fraction of *fake* candidates. These fake candidates are either completely

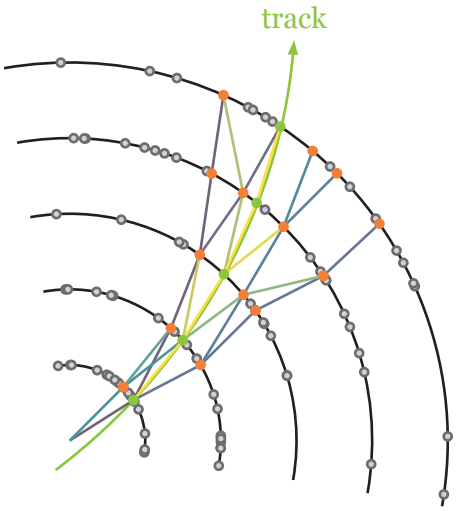


Figure 4.16.: Illustration of the way the CKF iteratively explores measurements from a seed outwards. Measurements are added successively, and can be shared between the resulting track candidates. Shown in green is a circular *real* trajectory.

combinatorial, or arise from real particle measurements with combinatorial additions. Aside from this, track candidates coming from a single seed necessarily share a common stem of measurements. Measurements can potentially also be shared between candidates from different seeds.

ATLAS tracking executes an ambiguity resolution algorithm, that attempts to filter out as many undesirable tracks as possible. This is implemented by means of a scoring function, that combines properties of the track parameters. Higher scores are correlated with a larger probability to be a desirable track candidate. A larger number of hits results in an increase in the score, as longer compatible hit chains are less likely to be random combinations. On the other hand, missing hits in sensors where a hit was expected negatively impact the score. Hits from the different detector subsystems are assigned different scores as well. The overall χ^2 value computed for the track candidate also plays a role. Candidates that share hits with other candidates are penalized. The Pixel cluster splitting algorithm, mentioned in section 4.5, can improve this situation, by allowing close by tracks to share measurements that were identified as merged. Another quantity is the measured particle p_{T}, which enters the score logarithmically, to give preference to tracks with large momenta. For tracks containing measurements with a substantial local χ^2_+ at the start or end of the trajectory, the algorithm will also attempt to remove these hits, and determine whether a refit without them yields a more favorable global χ^2. Finally, the output of the ambiguity resolution is a set of track candidates that contain an enhanced fraction of tracks from actual particles, while fake tracks are suppressed. They are passed into the final precision fit outlined in section 4.7, to extract the parameter estimate, and used for further aspects of reconstruction.

4.9. Vertex reconstruction

One very immediate use case for tracks is the reconstruction of interaction vertices in the event. These are needed for many aspects of higher-level reconstruction algorithms. Vertices are used in different ways. As shown in figure 4.17, a single event can have a primary vertex, usually associated with the hard-scatter event, as well as a number of secondary vertices. Typically, the primary vertex will be located in the luminous region, measuring about 10 µm in the transverse and 40 mm in Run 2. The reconstruction of secondary vertices can help with the identification of particles, for instance in the form of *b-tagging* jets originating from b-quarks. Here, a characteristic displaced vertex located inside a jet is a sign for a b-hadron decay. Additionally, pile-up interactions can produce vertices that are not associated with the interaction of interest. Signatures from pile-up interactions can be rejected, if they can be associated to a pile-up vertex.

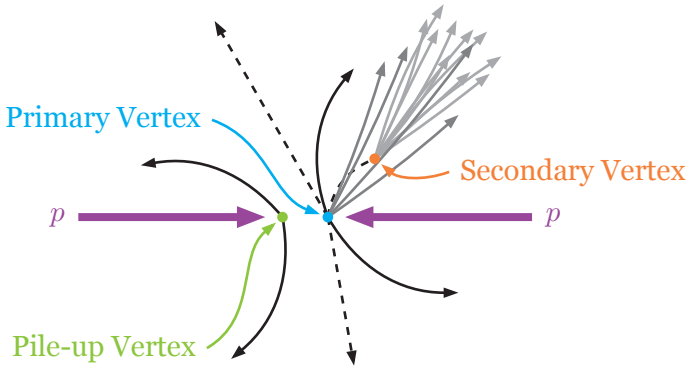


Figure 4.17.: Illustration of a set of three vertices in a proton-proton collision. A primary vertex and a secondary vertex are shown in addition to a pile-up vertex.

Vertex reconstruction can be divided into two stages: vertex finding and vertex fitting, in close analogy to the reconstruction of tracks themselves. Reconstruction begins with a seeding stage. Vertex seeding can be performed in a number of ways. Algorithms differ for primary and secondary vertex reconstruction. For primary vertex seeding, ATLAS runs an algorithm, which uses a histogram approach to cluster tracks in the z-axis. This is based on the assumption that primary vertices will align with the beam spot. Other approaches include the use of a Gaussian track density to identify regions with high track multiplicity as vertex seeds, a variation of which will be used starting in Run 3 [163]. For secondary vertexing, seeds are formed from pairs of reconstructed tracks, as the constraint to the beam spot does not apply.

With a set of vertex seeds determined, tracks compatible with the vertex are selected. Using a linearized approximation of the helical trajectories in the vicinity of the vertex [164], a fitting procedure based on the Kalman formalism can estimate the vertex parameters.

One issue with an approach like this is the fact that the assignment of tracks to vertices is not unambiguous. Improvements exist [165], where tracks that turn out to be a bad fit with the vertex in question are weighted down, avoiding degradation in the obtained result.

On the other hand, multi-vertex fit methods [163] can reassign such outlier tracks to a more compatible vertex, yielding improved results for both vertices.

4.10. Alignment of particle sensors

The geometry description introduced in section 4.4 models each sensor at an idealized position according to the detector design. In reality, construction has certain margins of error, and during operation, the geometry can move. This can occur due to external factors including the magnet system being toggled on and off, or changes in temperatures. Examples of misaligned sensors are shown in figure 4.18. Here, a layer of sensors is shown to shift in unison, or be compressed in one direction and expanded in another one. Finally, sensors are shown to feature a random shift with respect to their nominal position.

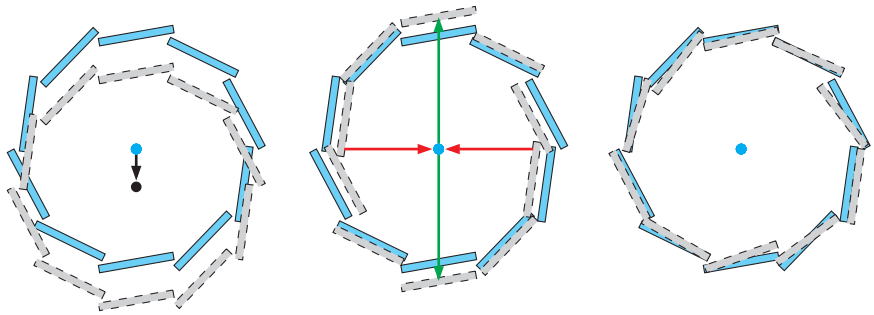


Figure 4.18.: Illustration of some of the ways sensors can become misaligned. (a) shows the shift of the full ring downwards, (b) shows a deformation of the ring, and (c) shows small random rotations and shifts of individual sensors.

If the sensor locations assumed by the track reconstruction are not correct with respect to the physical location of the hardware, such *misalignments* manifest themselves in the extracted track parameters. Both the impact parameter and the momentum resolution are affected, and can also introduce biases. Since the spatial resolution greatly influences many uses of track information, such as the identification of b-quarks, the degradation due to alignment is not acceptable.

This is especially true for misalignments of the innermost layer of the ATLAS Pixel detector, the IBL, which was added with the expressed goal of improving spatial resolution. During commissioning, it was discovered that the specific design of the IBL assembly results in an interface of materials with different thermal expansion properties in the stave. As a consequence, the staves deform [166]. The deformation is largest in the center of the staves, since they are mounted at the edges, and the central support structure allows movement along the azimuthal direction. An illustration of the effect of the IBL bowing is given in figure 4.19. On the left, a stave that would contain multiple sensors along z is shown. The bending occurs in the ϕ direction indicated in the figure. On the right is a representation of the cylindrical shape of the IBL as a whole. Here, the bowing is shown to be along the cover of the cylinder,

effectively *twisting* it in the center. The magnitude of the bowing depends on the operating conditions, but can reach values up to 10 µm/K [166].

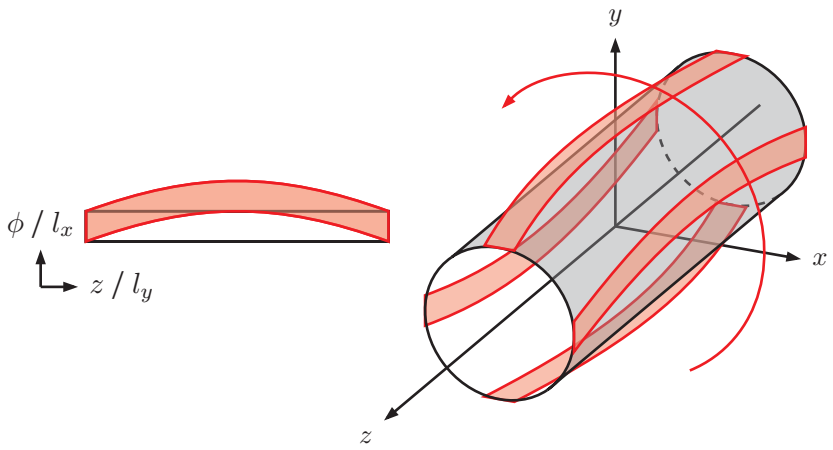


Figure 4.19.: Illustration of the bowing of the IBL. The left shows an individual stave, and how the bowing deforms it along the ϕ direction. On the right is the overall cylindrical shape of the IBL, and an indication of the deformation twist, alongside for exemplary staves.

In order to counteract the effects of the various misalignments, they first need to be measured precisely (see [167]). This is done by using a procedure that attempts to minimize an overall χ^2 value, assembled from the residuals of tracks in an event with respect to their constituent measurements. The minimization modifies the sensor locations and rotations according to their degrees of freedom. Overall, about 7.5×10^5 exist for the ATLAS tracking detectors. Care has to be taken to use external constraints to suppress biases from alignment modifications that leave the χ^2 value invariant.

Multiple stages of the measurement procedure exist, where parts of the detectors are assumed not to move to serve as a reference frame for other parts of the detector that are under interrogation. Finally, the output of the procedure are alignment corrections that need to be applied to the nominal tracking geometry, in order to reproduce the measured misalignment during reconstruction. The corrections are categorized into three levels: Level 1 includes adjustments to the combined structures of the IBL, Pixel, SCT endcaps and TRT barrel and endcaps, separately. The SCT barrel is fixed at Level 1. Level 2 allows variations of the combined location and rotation of the silicon layers. For the TRT, the individual endcap wheels can move, as do the TRT barrel modules. They assume the silicon detectors to be fixed. Finally, at Level 3, individual silicon sensors are varied, as are the straws of the TRT.

Of course, since the misalignment varies over time, the corrections are also time-dependent. The duration in time, during which the alignment is assumed to be constant is referred to as an Interval Of Validity (IOV). The effective IOVs differ for the various alignment levels. Movement of the macro structures of the ID are usually due to external factors, such as the magnets being powercycled, which occur at a low frequency. Depending on the exact mechanism of effect, temperature can have varying impact. One particular effect is the Pixel detector rising at the beginning of data taking, before settling down and lowering itself

relative to the SCT. This is believed to be due to coolant temperature changing. At the highest frequency is the alignment of the IBL. The temperature scale that affects its alignment is so low, that the readout itself causes enough temperature variation to affect the bowing strength. As a consequence the IBL staves are approximately straight at the beginning of a run, and begin to bow as the run continues. To be able to compensate for this, alignment constants can be calculated for IOVs as short as one LB corresponding to at least 60 s.

The technical application of time-dependent misalignments in a concurrent fashion as part of the A Common Tracking Software (ACTS) project in the ATLAS software is the focus of section 7.7.

4.11. Summary

Chapter 4 concerned itself with the reconstruction of particle tracks, with a particular focus on methods used in the ATLAS experiment. Charged particle detection forms the basis of track reconstruction, and various methods to achieve it exist. Using pattern recognition, detector signals can be converted into spatial representation. In the *seeding and track following* approach, track candidates are formed from measurements, and then extended with additional measurements based on their compatibility. For the final parameter estimation, the trajectories are fitted using various approaches. One important ingredient is the ability to predict particle trajectories based on their current parameters. This is achieved using numerical integration techniques, in the presence of magnetic fields.

Tracking performance is largely influenced by the amount of passive material that particles encounter when traveling through the detector. Minimizing it is therefore a good strategy to improve resolutions and limit uncertainties. Intrinsic effects, like the segmentation of sensitive elements, also plays an important role. Precise knowledge of the magnetic fields used to induce bending is critical for precision of particle propagation. Finally, misalignment of sensitive elements needs to be carefully determined and corrected for, to maintain high accuracy.

5. HL-LHC and challenges of the High-Luminosity era

5.1. The High-Luminosity-LHC

Since 2008, CERN operates the LHC. In the successful data-taking campaigns from 2009-2013 and 2015-2018, dubbed *Run 1* and *Run 2*, proton-proton collisions amounting to about 185 fb^{-1} of integrated luminosity were delivered. Out of this, a total of 163.9 fb^{-1} of was recorded by ATLAS and deemed good for physics, at center-of-mass energies ranging from $\sqrt{s} = 7\,\text{TeV}$ to 13 TeV.

For searches for new physics and highest precision measurements, the amount of available statistics in the used datasets can still present a limiting factor. Without any clear indication of new physics observed or on the horizon, the focus of proton-proton physics shifts toward these precision measurements, to either further refine knowledge of the Standard Model (SM), or find new physics in any discrepancies.

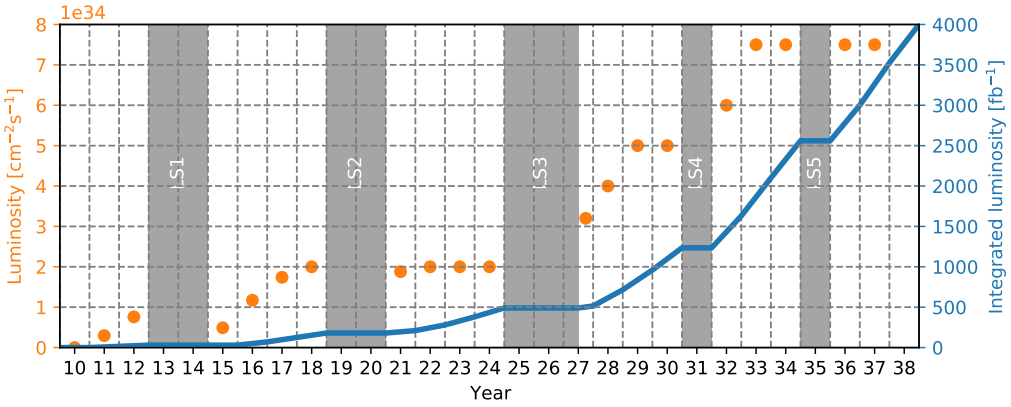


Figure 5.1.: Real and planned evolution of the instantaneous and integrated luminosity as a function of time. Data points taken from [168].

To this end, CERN launched the High Luminosity Large Hadron Collider (HL-LHC) project [169], an upgrade to the LHC. It intends to deliver unprecedented amounts of data by means of an increased instantaneous luminosity. A host of new technical developments are underway to prepare or replace most parts of the LHC to be able to deliver this beam intensity reliably. One result of these efforts is an increased number of interactions per proton-proton bunch crossing, commonly referred to as pile-up. In Run 2, the mean number of interactions per bunch crossing amounted to $\langle\mu\rangle = 33.7$, with the instantaneous luminosity peaking at $2.1 \times 10^{34}\,\text{cm}^{-2}\,\text{s}^{-1}$.

HL-LHC is designed to achieve an instantaneous luminosity of $7.5 \times 10^{34}\,\mathrm{cm}^{-2}\,\mathrm{s}^{-1}$, which will correspond to $\langle\mu\rangle$ of up to ~ 200. The planned LHC schedule and luminosity program is shown in figure 5.1, indicating that the integrated luminosity planned to be recorded after LS3 grows at a much accelerated rate. A target integrated luminosity in 2038 of $4\,\mathrm{ab}^{-1}$ is foreseen.

An increase in the instantaneous luminosity results in an improved rate at which events of interest occur. However, this increase in instantaneous luminosity also comes at a price. Firstly, while an increase in the rate of events of interest is desirable, if all or a significant portion of these events are to be analyzed, they need to be recorded. Thereby, additional strain is put on the trigger and data acquisition chains of experiments. Secondly, the number of interactions per proton-proton bunch crossing affects the operation of the detector hardware. For example, sensitive elements experience increased occupancy, which can affect their efficiency. Due to the elevated flux of ionizing radiation, the components also undergo more rapid degradation, which needs to be carefully monitored and mitigated. Thirdly, pile-up also has an impact on the processing and analysis of the recorded events. In many cases, background activity in the detectors, arising from larger pile-up, complicates reconstruction algorithms, and affects their efficiency, both in terms of computing and physics performance. Even though the increase of a factor of about six in $\langle\mu\rangle$ might appear modest, many algorithms scale super-linearly with the mean number of interactions. This makes the high-luminosity era a challenge in terms of computing as well.

To adapt to these challenges, the LHC experiments are conducting ambitious upgrade programs. One component of these upgrades is on the detector hardware. For example, ATLAS is working on a complete replacement of the Inner Detector with the Inner Tracker (ITk), which is the focus of section 5.3.1. Additionally, in the endcap regions, a High-Granularity Timing Detector [170] will be introduced. This subsystem will enable association of timing measurements with tracks extended from the ITk, yielding additional pile-up rejection power. ATLAS is also upgrading large parts of the trigger hardware and infrastructure, to cope with increased interaction rates. Parts of the muon systems are replaced to enhance muon trigger capabilities, in addition to the New Small Wheel [103] upgrade being completed. On the software and computing side, substantial upgrade efforts are undertaken as well, and are the topic of section 5.2.

5.2. Software and computing challenges

For the upcoming high-luminosity environment, remaining within CPU and storage resource budgets will become critical for experiments at the HL-LHC. While CPU consumption will become approximately constant once instantaneous luminosity reaches its design value, disk usage will rise throughout the operation of the machine. ATLAS is mounting a mitigation strategy, outlined in [168].

Storage bottlenecks are addressed with new analysis formats, and the switch to tape storage where possible. Clearly, these improvements in storage efficiency also require adaptations in the associated software. Work is underway to implement slimmed down analysis formats

DAOD_PHYS and DAOD_PHYSLITE, which target 50 kB and 10 kB per event, respectively. As a comparison, the target size per event for the main xAOD is around 100 kB. Both of these formats are intended to replace the currently numerous dedicated data formats used in analyses, by providing the majority of required data and quantities needed for most of them. The intention is to reduce the overall storage consumed by all analysis formats in combination. They also make use of lossy compression, to bring numerical precision in line with actual measurement uncertainties. The DAOD_PHYSLITE also contains calibrated objects, avoiding the need to store information needed to perform these calibrations. Which impact these new formats will have on overall storage usage is contingent on adoption in analyses.

To address the usage of CPU resources, optimizations and improvements occur in many domains, including event generation and simulation. Event generation amounts to roughly 15 % of CPU resources, while the simulation of detector response takes up about 40 %. Event reconstruction, which takes up a large fraction of the remaining CPU resources, is also an area of focus. One key observation is the apparent end of compute performance increasing at constant hardware prices over time. Manufacturers mitigate this by scaling up the number of processing cores included in a CPU. On the other hand, memory prices stagnating mean that the available memory per core is likely not to increase. It is therefore paramount to make optimal use of the available memory in software.

The ATLAS software, Athena [109], is set up to execute on multiple CPU cores concurrently, by splitting up execution into multiple processes. These processes are created from an initial process by means of a *fork*. After *forking*, already allocated memory of the original process is accessible by all newly created processes. The operating system ensures that whenever memory is written to, the corresponding data is copied into process-specific memory allocations[1]. Athena employs some strategies to maximize the amount of shared read-only memory usable in this way.

To overcome the limits of this method of sharing memory, multiple *threads* of execution can be launched within a single process. Such threads are able to share the entire memory of the process for both reading and writing. While this enables the possibility of more efficient memory usage, great care has to be taken to ensure no memory corruption occurs due to concurrent read and write operations of the same memory locations.

ATLAS is in the process of adapting its software to efficiently use multi-threading in the *AthenaMT* project [171]. Here, most components of the event processing chain are executed by a scheduler, which distributes the work across a number of threads, potentially out of order. By explicit declaration of data-flows, the scheduler can ensure that components depending on the output of other components run in the appropriate order. Such an execution strategy is illustrated in figure 5.2, where two threads run components (algorithms) on three different events. The execution within an event does not necessarily occur sequentially, data dependencies are respected to ensure correct results. Individual components need to be designed or migrated to be thread-safe, such that they can be safely executed in this fashion. In this execution environment, components can then leverage in-process shared memory.

[1] Copy-on-write

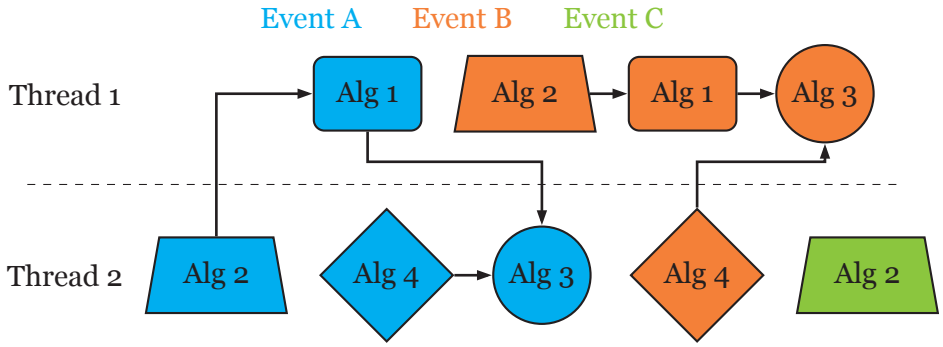


Figure 5.2.: Illustration of the concept of multiple threads executing events concurrently. Two threads are shown to execute on three events, drawn in different colors. The data flow between algorithms is always the same, the scheduling can switch between events, however.

The migration effort is making good progress, but has yet to be completed.

Aside from the correctness of processing and acceptable memory requirements, the time needed for event processing also rises considerably. Given the much larger expected number of mean interactions per bunch crossing, most algorithms take significantly more time to complete. One particularly susceptible domain is track reconstruction, whose runtime requirements scale with event activity. Procedures such as seed formation and subsequent track finding are heavily affected by increased combinatorics. Therefore algorithmic improvements are needed as well, to bring CPU consumption in line with available resources.

In the track reconstruction domain, the cross-experiment tracking toolkit ACTS will be leveraged in ATLAS to address both thread-safety of the track reconstruction software suite, as well as its runtime performance. ACTS is one of primary areas of work in this thesis, and is discussed in detail in part II. Other efforts to address the resource consumption of tracking in the upgraded ATLAS experiment are outlined in section 5.3.2.

5.3. Tracking in ATLAS at the HL-LHC

5.3.1. ATLAS Phase-2 Upgrade Inner Tracker

The increased pile-up environment expected at the HL-LHC motivates the complete replacement of the ATLAS ID. Eschewing the usage of a transition radiation tracker, the new Inner Tracker (ITk) [172–174] consists entirely of silicon based detectors. An overview of the new layout is given in figure 5.3. As is visible there, the general design philosophy remains: The detectors follow the pattern of cylindrical barrels with disks in the endcaps to form a hermetic detector volume. Primary goals of the upgrade are to maintain track parameter resolutions even in the presence of high pile-up and extend coverage to $|\eta| < 4$. Stated target resolutions are $\sigma(q/p_\mathrm{T}) < 0.3\,\mathrm{TeV}^{-1}$, $\sigma(d_0) < 8\,\mu\mathrm{m}$ and $\sigma(z_0) < 50\,\mu\mathrm{m}$ at the limit of particles with infinite momentum in the central part of the detector. At the same time, the components should be robust enough to radiation damage to cope with the harsh environment expected at

the HL-LHC.

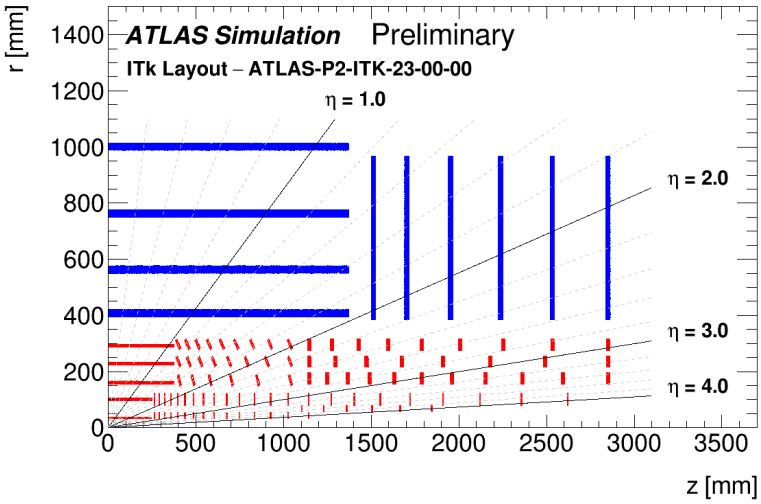


Figure 5.3.: View of the ITk layout in the rz-plane. It shows the pixel layers in red, with cylindrical barrel layers in the center, and inclined and conventional disk layers in the endcaps. In blue the strip layers are visible, with four cylindrical barrel layers and six endcap disks. Figure taken from [175].

The upgraded Pixel detector will feature five barrel layers with radii between 34 mm and 291 mm in the flat central region. A noteworthy feature is the split in an *inner* and *outer* Pixel system. It allows the replacement of the inner system without touching the outer one, in case this is needed due to radiation damage during operation. The inner system consists of the two innermost layers across barrel and endcaps, seen at $r < 150$ mm in the picture. The other three layers belong to the outer system. Additionally, the endcaps disks in the inner Pixel system are coupled between the innermost and next-to-innermost rings in r. This allows a more optimal routing of services and reduce material. The outer three barrel layers are complemented with a number of endcap disks whose sensors are inclined in the longitudinal plane. The inclination allows for a reduction in material a particle encounters, and requires less silicon surface to cover a larger η range. They are visible between about 400 mm $< z <$ 1020 mm and 130 mm $< r <$ 300 mm in figure 5.3. Aside from these, the new Pixel system includes a number of more conventional endcap rings. In total, this extends the Pixel coverage to $|\eta| = 4$, as can be seen in figure 5.3. Pixel sizes range from 25 µm × 100 µm to 50 µm × 50 µm. Overall, ITk will feature about 9000 pixel sensors.

For the strip system, the number of barrel layers remains at four at radii from 399 mm to 1000 mm. In the endcaps a total of six rings ranging from 1512 mm $< z <$ 2850 mm are shown in the schematic. Longitudinal coverage of the new strip system will extend up to $|\eta| = 2.7$. Barrel sensors are rectangular, with a strip pitch of 75.5 µm. A specific feature of the upgraded strip endcaps is a new module design, that uses a custom shape with built-in stereo angle of 40 mrad. An in-depth discussion of aspects of this new sensor type and its implementation in software can be found in section 7.3.

The overarching goal of this layout of the detector is to achieve upwards of nine precision

measurements for any charged particle in excess of $p_{\mathrm{T}} > 1\,\mathrm{GeV}$. At the same time, careful optimization of the distribution of support structures and auxiliary devices allows a reduction in material that particles need to traverse before reaching the minimum required hit threshold. Studies documented in [174] show that the minimum hit requirement can be comfortably achieved across the entire η coverage of the ITk. Aside from this, track parameter resolutions change with respect to the ID in Run 2 conditions. The transverse impact parameter resolution is expected to match or slightly degrade, depending on p_{T}. Transverse impact parameter resolution is expected to improve, mainly due to the reduced pixel sized in that direction. Momentum resolution is generally improved, due to reduced material and higher precision compared to the TRT. Angular resolutions are slightly improved. Using an optimized deployment of different readout strategies, in combination with choices made in the layout of the sensors, readout rates and sensor occupancy can be constrained to tolerable levels even at $\langle\mu\rangle \sim 200$.

5.3.2. Track reconstruction with the ITk

To cope with changing requirements, experiment software needs to be equipped to handle the task of reconstructing and otherwise processing of recorded data. The existing tracking infrastructure in ATLAS is tuned to handle pile-up of up to about $\langle\mu\rangle \sim 60$, which was the approximate maximum during Run 2. Figure 5.4 shows the scaling of CPU time used in track reconstruction as a function of pile-up, measured in MC simulation. Runtime performance is shown in multiples of the HS06 [176] benchmark score obtained by the processor these studies were run on, for better comparability. In a dashed line, the scaling of the existing Run 2 reconstruction are shown for values up to $\langle\mu\rangle = 60$. The scaling is quite drastic, as runtime increases more than five-fold between $\langle\mu\rangle = 20$ and $\langle\mu\rangle = 60$. This benchmark is based on the Run 2 detector hardware and geometry.

Shown in black is the scaling of the *default* ITk reconstruction. While the software and strategy for reconstruction is very similar, this benchmark is run on the upgraded ITk geometry. It is clear that the move to an all-silicon detector, and removal of the TRT has a beneficial impact on the scaling. Aside from this, the fact that seed pointing resolution is much improved allows for far more stringent selection criteria at the seeding stage. This, in turn, reduces the impact of combinatorics between space points substantially. Other improvements, including the minimization of the effective gap between silicon layers at the boundary between the Pixel and the Strip detector, also contribute to an already significantly more favorable scaling.

To further improve this scaling, additional optimizations can be made. In order to enable the potential usage of an online software track trigger, as an alternative to the planned Hardware Track Trigger (HTT) [178], these optimizations need to be relatively aggressive to achieve reasonable runtime performance. Such an optimization strategy is described and evaluated in [177]. Here, the ambiguity resolution stage is dropped completely, since it was identified as the largest consumer of CPU time. Additionally, the fit result from the track finding stage is used directly, without performing a precision refit. To compensate the about a factor of two larger number of track candidates, track selection criteria in the track finding stage were

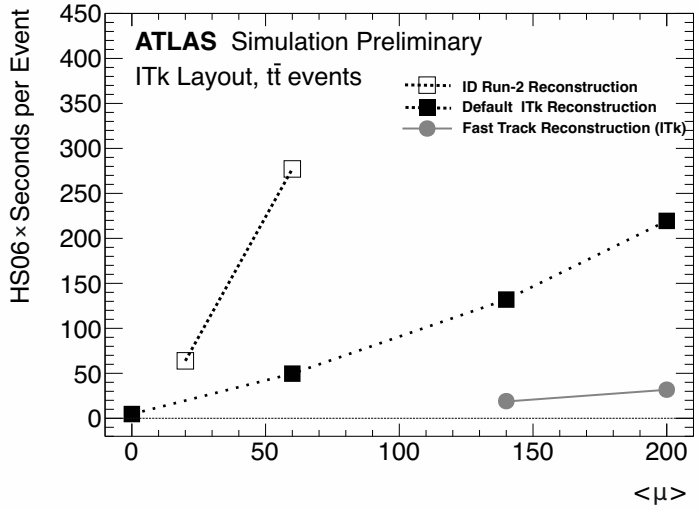


Figure 5.4.: CPU performance scaling as a function of $\langle\mu\rangle$. Shown are the ID reconstruction under Run 2 conditions, the *default* ITk reconstruction and a fast variant of it. Figure taken from [177].

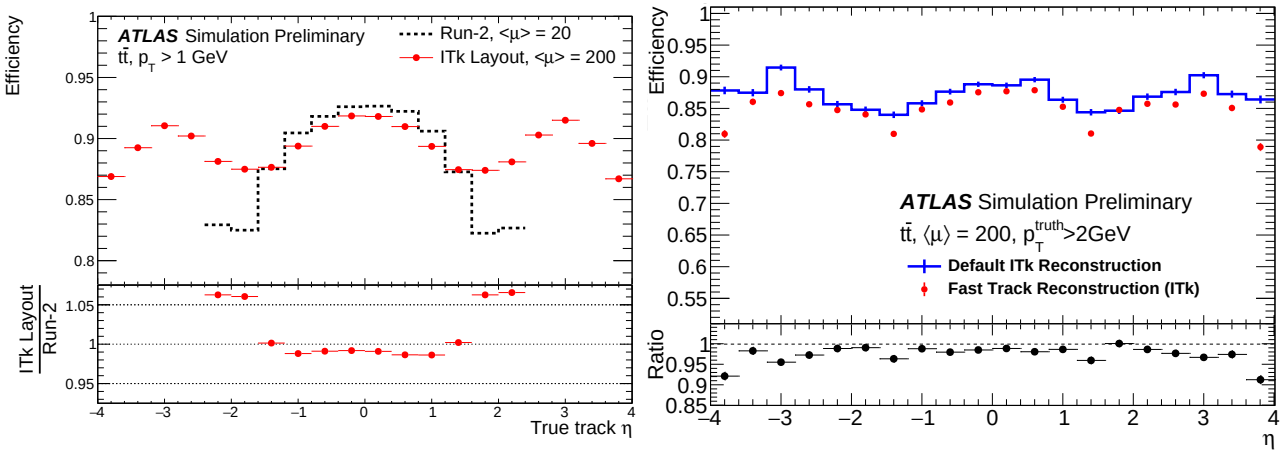
tightened. Changes for this track selection include an increase of the minimum p_{T} threshold from 900 MeV to 1 GeV, increasing the minimum required silicon hits for a track from seven to nine, and tightening the longitudinal impact parameter cut to $|z_0|$ to < 15 mm from 20 mm in the central detector region at $|\eta| < 2.0$. This last change is motivated by the reduced size of the interaction area in the HL-LHC. At larger $|\eta|$, the momentum and hit requirements are tuned appropriately. An overview over these requirements and two additional hit-based criteria for the η regions for *fast* and *default* configurations can be found in table 5.1.

Requirement	$\lvert\eta\rvert < 2.0$		$2.0 < \lvert\eta\rvert < 2.6$		$2.6 < \lvert\eta\rvert < 4.0$	
	fast	default	fast	default	fast	default
Pixel + Strip hits	≥ 9	≥ 7	≥ 8	≥ 7	≥ 7	≥ 7
unique hits	≥ 7	≥ 1	≥ 6	≥ 1	≥ 5	≥ 1
shared hits	≤ 2	—	≤ 2	—	≤ 2	—
p_{T} [MeV]	> 1000	> 900	> 400	> 400	> 400	> 400
$\lvert z_0\rvert$ [cm]	≤ 15	≤ 20	≤ 15	≤ 20	≤ 15	≤ 20

Table 5.1.: Table showing the cut values for three $|\eta|$ regions, tuned for the *fast* reconstruction, and the corresponding *default* values. Numbers taken from [177].

Drawn as filled circles in figure 5.4 is the runtime scaling of this fast tracking variant for $\langle\mu\rangle = 140$ and 200. It is clear that this scaling is much flatter than both Run 2 and *default* ITk versions. At the same time, tracking performance is only slightly degraded, as can be seen in figure 5.5. Shown in figure 5.5a is a comparison of the track reconstruction efficiency between the Run 2 reconstruction at $\langle\mu\rangle = 20$, and the ITk *default* reconstruction at $\langle\mu\rangle = 200$. The efficiency is measured in $\bar{t}t$-events, with a prior truth cut of $p_{\mathrm{T}} > 1$ GeV. The greatly increased coverage of the ITk in η is clearly visible. While the Run 2 efficiency drops at $|\eta| \sim 1.6$ and ends at $|\eta| = 2.5$, the ITk efficiency remains above 86 % up to the full $|\eta| = 4$.

Figure 5.5b shows a comparison of the same efficiency between the *default* reconstruction



(a) (b)

Figure 5.5.: Track reconstruction efficiency as a function of true particle η, measured in $\bar{t}t$-events. (a) shows a comparison between the Run 2 reconstruction at $\langle\mu\rangle = 20$ and the reconstruction in the ITk at $\langle\mu\rangle = 200$ [174]. (b) compares the same ITk reconstruction with a tuned fast variant. [177]

from before[2], and the new fast variant. It can be observed that the fast variant only present slightly worse reconstruction efficiencies across the entire η range. As documented in more detail in [177], other quantities show similarly small degradations in performance. It is therefore expected that CPU scaling of the track reconstruction can be largely brought to acceptable levels in this fashion.

[2]Differences between the *default* ITk reconstruction in figures 5.5a and 5.5b can be attributed to different truth cuts $p_T > 1$ GeV and $p_T > 2$ GeV, differing geometry layouts and software versions.

Part II.

Development and improvement of track reconstruction software

6. The ACTS project

A large part of the work conducted in the process of this thesis was in the context of the ACTS (*A Common Tracking Software* [179]) project. ACTS is an experiment independent software framework for track reconstruction. In this chapter, a general introduction to the framework is given. Major features, design goals and decisions are introduced, discussed and motivated. Thread-safety, a selection of specific components and development practices is the area of focus. The software is under active development by a number of contributors. Therefore, this chapter and the following one cover the status of the project during summer 2020, where at the time of writing, the most recent version is `v0.25.0`.

6.1. Overview

6.1.1. Origins and objectives

The ACTS project started as an attempt at rewriting the tracking software used for reconstruction in ATLAS [138] in an experiment independent way. The ATLAS tracking software has been used extensively in the data taking during Run 1 and Run 2 of the LHC. It was found to be generally performing well, both from a physics performance and from a CPU performance perspective. Since work on the ATLAS tracking software started before the first run of the LHC, the software grew more or less organically to about a million lines of code. Over time, adjustments to changing requirements were made, as well as new features built when they became necessary. In a number of refactoring campaigns, the ATLAS tracking software was restructured, for example replacing the `CLHEP` [180] linear algebra with `Eigen` [181]. However, these projects required major efforts, due to interactions with a software that was already used in a production context.

As with any large software project, this leads to the software becoming less and less structured, with clarity of code and maintainability at a lower priority in favor of working code. While this is mostly true for the entire ATLAS software, in particular the tracking software suffered from contributors moving on, leaving some parts essentially unmaintained. For reasons outlined in section 3.3.2, the ATLAS software is moving to a new concurrent execution model. Large parts of the code have to be made compatible with this change, which is a daunting task, given the size and complexity of the code base. Approaches and incremental developments have to be tested and verified, before committing to a global migration. To facilitate the process, the tracking code was extracted from the ATLAS software framework, and published under an open-source license[1]. This extracted code formed the basis of the

[1]The ATLAS software Athena has since also been released as open-source software.

ACTS project, and has seen extensive development on top of it. The decoupled ACTS code can be used as an environment for prototyping approaches to thread-safety, that is essential for concurrent execution. At the time of writing, the code base is roughly a tenth of the size of the ATLAS tracking software, at 140 000 lines of code. To leverage improvements achieved in the ACTS codebase, an eventual reintegration and usage of ACTS components was foreseen from the beginning, and is discussed in chapter 7.

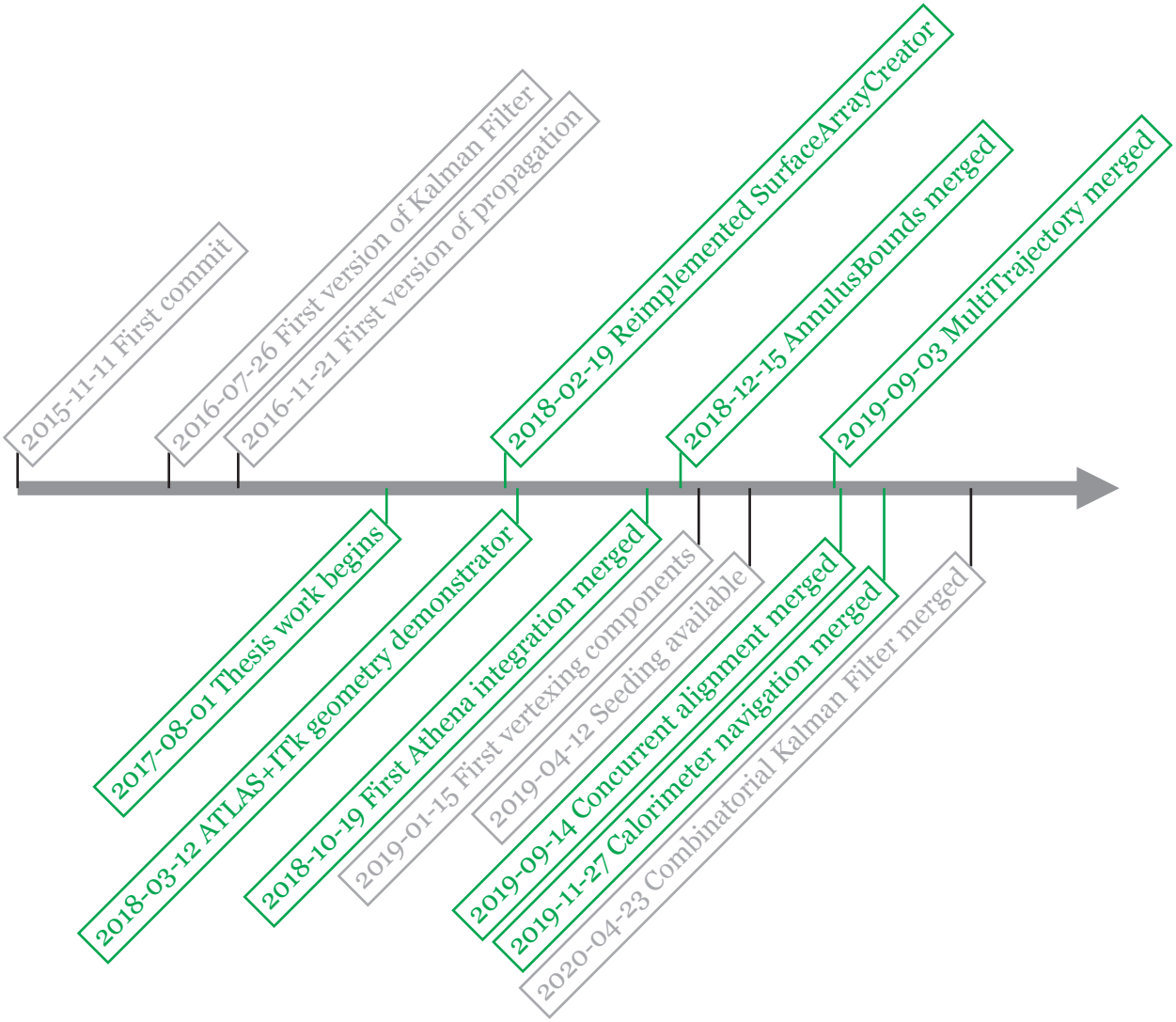


Figure 6.1.: Overview of the timeline of development of ACTS from 11. Nov 2015 until 01. Jan 2021. Shown are selected contributions that were part of this thesis, and other contributions.

6.1.2. Strategy and goals

The strategy for the project is to keep the algorithms and techniques of the ATLAS tracking software that have proven to work well, and improve on other aspects. Links in the reconstruction chain consume inputs and provide outputs to others. In ATLAS, large parts of the

chain are tightly coupled, requiring adjustments to a large amount of client code for every change. Prototyping of new developments thus increases in complexity.

Given the extraction from the ATLAS code motivated above, the software becomes technically independent of the experiment itself. Dependencies on other parts of the ATLAS software that prevented a standalone build were removed. Parts of the code that were obsolete in a standalone environment were removed as well. In addition, ATLAS-specific assumptions and shortcuts were removed where possible. Although assumptions on the general structure and approach to track reconstruction remain the same, re-implemented algorithms are designed to be experiment independent. This enables running tests and examples in smaller executables, which allows for shorter development iteration cycles.

An additional benefit is that the software can now applied to other experiments with relative ease. In fact, ACTS is intended to provide a toolbox of track reconstruction algorithms and supporting libraries that can be applied to other experiments. Providing a turnkey software for track reconstruction is out of scope for the ACTS project, however. Instead, it is intended to be used to assemble experiment-specific *applications*, that perform track reconstruction from start to finish. Generic ACTS components can be supplemented with more specific components depending on the use case. This is possible, because a large fraction of the algorithmic code is decoupled from the mathematical operations.

Apart from these benefits, ACTS provides an open-source platform for development of new algorithms or approaches. By pooling resources with contributors from other experiments, transfer of knowledge which was previously experiment-specific can be enabled. If algorithms are implemented in a common environment and on top of the same supporting architecture, like a common event data model, they can also be compared and benchmarked against each other more easily. ACTS also includes a library for fast simulation of particle tracks, taking into account interaction of particles with the detector material, energy loss, digitization, among others. This library has been used to generate a generic dataset of simulated tracks that was used as the input for the TrackML challenge [182]. The dataset has since been used in a variety of other research and development projects, demonstrating the community benefit achievable.

6.1.3. Code maintainability

An important factor in software development in general is the ability to continue development and support for a software. Depending on the complexity of the code, the difficulty of finding contributors and the time until they can make contributions can vary drastically. This is described by the concept of maintainability. Wherever software is used for many years, this becomes crucial since changes need to be incorporated and the software needs to be adapted to changing environments, as is the case for software in HEP. To improve maintainability of the code extracted from the ATLAS tracking software, several measures were taken.

The first measure is extensive usage of the `Eigen` [181] library for linear algebra. Aside from the excellent runtime performance characteristics of the library, the use of operator-overloads in C++ enables expression of matrix operations in a very readable fashion. This is illustrated

$$M_{4\times4}$$
$$N_{4\times4} = \mathbb{1}_4$$
$$X_{4\times4} = M_{4\times4}^{-1} \times N_{4\times4}^{T}$$

```
Eigen::Matrix4f M{}; // ...
Eigen::Matrix4f N = Eigen::Matrix4f::Identity();
Eigen::Matrix4f X = M.inverse() * N.transpose();
```

$$\vec{v} = (1, 2, 3, 4)^T$$

```
Eigen::Vector4f v{};
v << 1, 2, 3, 4;
```

$$\vec{w} = X_{4\times4} \cdot \vec{v}$$
$$\vec{u} = 5\vec{w}$$
$$f = \vec{u} \cdot \vec{v}$$

```
Eigen::Vector4f w = X * v;
Eigen::Vector4f u = w * 5; // multiply with scalar
float f = u * v; // dot product
```

Listing 6.1.: Some examples how linear algebra expressions translate to `Eigen`. Shown are matrix multiplication, vector assignment, matrix-vector and scalar-vector multiplication, as well as the dot product of two vectors.

in listing 6.1. Here, various linear algebra operations are listed, and how they can be expressed using `Eigen`. The syntax is concise, and maps nearly directly to mathematical notation. While the ATLAS tracking software already uses `Eigen`, a sizeable number of parts are implemented with hand-written procedures.

The benefits of using the library become especially apparent in code relating to the propagation of particle trajectories (see section 4.3). The ATLAS propagation implements an algorithm based on Runge-Kutta-Nyström [143, 144], and contains code which operates on manually allocated contiguous memory. While this certainly makes the performance characteristics virtually optimal, it results in code that contains opaque index manipulations and calculations. Tracing down the higher-level mathematical operations occurring in the code becomes drastically more difficult. An example of this can be found in listing 6.2 for a central part of the numerical particle trajectory integration, the update of the trajectory position and direction. Listing 6.2a is taken from the `AtlasStepper`, a nearly identical copy of the ATLAS Runge-Kutta-Nyström implementation, that was only adjusted to conform to the different interface. It is used to validate the numerical results of the `EigenStepper`, whose equivalent calculation is shown in listing 6.2b. Mathematically, the update is done using the following expressions [144]:

$$\begin{aligned} \vec{T}_{n+1} &= \vec{T}_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4) \\ \vec{r}_{n+1} &= \vec{r}_n + h\vec{T}_n + \frac{h^2}{6}(k_1 + k_2 + k_3). \end{aligned} \tag{6.1}$$

Here, $\vec{r}_n$ and $\vec{T}_n$ are the position and direction, h is the step size, and k_i correspond to the four points at which the integrated function is evaluated. Comparing listing 6.2b and equation (6.1) reveals a very clear relationship. The same cannot be said for listing 6.2a, the code manually indexes into arrays, and uses raw pointers to them. The operations required are very verbose,

```
double* R = &(state.stepping.pVector[0]);  // position
double* A = &(state.stepping.pVector[4]);  // direction
// Calculate RK step. {A,B}{0,1,2,3,4,5,6}, S3 are result/temporary

// update direction
A[0] = 2. * A3 + (A0 + A5 + A6);
A[1] = 2. * B3 + (B0 + B5 + B6);
A[2] = 2. * C3 + (C0 + C5 + C6);
double D = (A[0] * A[0] + A[1] * A[1]) + (A[2] * A[2] - 9.);
double Sl = 2. / h;
D = (1. / 3.) - ((1. / 648.) * D) * (12. - D);
// update position
R[0] += (A2 + A3 + A4) * S3;
R[1] += (B2 + B3 + B4) * S3;
R[2] += (C2 + C3 + C4) * S3;
// finalize direction
A[0] *= D;
A[1] *= D;
A[2] *= D;
```

(a)

```
state.stepping.pos +=
  h * state.stepping.dir + h2 / 6. * (sd.k1 + sd.k2 + sd.k3);
state.stepping.dir += h / 6. * (sd.k1 + 2. * (sd.k2 + sd.k3) + sd.k4);
state.stepping.dir /= state.stepping.dir.norm();
```

(b)

Listing 6.2.: Listing showing a crucial part of the Runge-Kutta-Nyström integration code in both the `AtlasStepper` (a)[2], and `EigenStepper` (b)[3] . The former is translation of the ATLAS propagator to ACTS with minimal changes, and contains hand-written vector operations. The latter uses the `Eigen` library to express mathematical operations. Both pieces of code calculate the updated position and direction after a Runge-Kutta-Nyström step.

and hard to follow. Lines like

```
D = (1. / 3.) - ((1. / 648.) * D) * (12. - D);
```

are the result of manual optimization by computing values ahead-of-time, and inserting them into the code as literals. In ACTS, such values are either derived automatically, or explicitly marked as tunable parameters of the algorithm.

`Eigen` also has the ability to specify the dimensions of matrices and vectors at compile-time. By doing that, it can avoid dynamic memory allocations, and potentially select the most appropriate algorithms for certain types of operations depending on the matrix sizes ahead of time. These optimizations can result in execution times reduced by 30 %-50 % compared to dynamic operations.

Another aspect of the library is a concept called *expression templates*. Rather than automatically evaluating the numerical result of every operation eagerly, the computation will

only be performed when necessary. It is implemented using template metaprogramming, and can avoid performing unnecessary intermediate calculations. Conversely, it can also be detrimental to runtime performance if not used carefully, as intermediate results might be calculated multiple times instead of being reused. An example for expression templates is found in listing 6.3. Here, an expression is constructed in the third line, but not evaluated until the multiplication in the last line. No intermediate result is calculated, thereby avoiding overhead.

```
Eigen::Matrix4f m1 = getMatrix();
Eigen::Matrix4f m2 = getMatrix();
auto m3 = m1 * m2; // no calculation is performed,
                   // the type captures the operation
Eigen::Matrix4f m4 = m3 * 42; // calculation is performed on assignment
```

Listing 6.3.: Illustration of the way `Eigen` captures expressions using compile-time types, and lazily executes calculations when necessary.

`Eigen` is widely used by projects such as TensorFlow [183], and was originally developed for the KDE Linux desktop environment [184], indicating that it is a solid and maintained library.

A second measure is removal of parameters appearing in code without context or explanation. These typically result from manually pre-calculating inputs to a calculation, or hand-tuning a free parameter of a procedure. Instances of pre-calculated parameters were replaced either with zero- or low-cost automatic calculations. Hand-tuned inputs were either replaced with automatic tuning based on more obvious inputs, or they were clearly marked and made configurable.

Apart from the aforementioned `Eigen` library, the only other required dependency is `Boost` [185]. It is used to provide features like command line options for configurable executables, as well as a framework for unit testing, which will be discussed later. Reliance on `Boost` for other aspects has been continuously reduced during the work for this thesis, in favor of replacements from the C++ standard library. Limiting reliance on external libraries avoids depending on their continued development. Any other external library dependencies are purely optional, and can be configured individually, as will be discussed shortly.

6.2. Thread-safety

As discussed in section 5.2, the Athena software framework [109] will move to a multi-threaded parallel execution model [171] to meet the resource and performance requirements posed by Run 4 conditions. To enable this execution model, the entire software needs to be made thread-safe, as was discussed previously in section 5.2. Due to the size of the software, this is a major undertaking. Thread-safety, in this context, can be divided into two separate levels: The first level marks code that *works* when run in multiple threads, which means the absence of crashes or regressions. While this is a necessary step, ideally, the implementation makes optimal use of the concurrent execution. This optimization marks the second level.

Implementing this second level is a major effort in its own right, and becomes more difficult in an environment with many dependencies. The ATLAS tracking software features deep call-chains and many stateful components. As stateful components require synchronization to be thread-safe, and deep call chains make it difficult to determine where the state is located, this poses a challenge.

There is no single approach to achieving optimal thread-safety. ACTS is designed with thread-safety in mind, such that the components can be used directly in multi-threaded software. Due to this, retrofitting existing code with synchronization measures to make it thread-safe can be avoided, as is required in the ATLAS software. In ACTS, no assumptions about the specific threading model or backend are made.

```
struct Tool {
  // "Bad": Implicit mutable state
  double m_value;
  void hiddenState(double x) {
    m_value = x + 5;
  }

  // "Worse": Mutable state masked by 'const'
  mutable double m_value2;
  void mutableKeyword(double x) const {
    m_value2 = x + 5;
  }

  // "Worse": Cast constness away
  double m_value3;
  void constCast(double x) const {
    const_cast<Tool*>(this)->m_value3 = x + 5;
  }

  // "Good": Explicit mutable state as reference argument
  struct State {
    double value;
  };
  void doSomething(State& state, double x) const {
    state.value = x + 5; // mutable state, constant method
  }
};
```

Listing 6.4.: Examples of a mutable member in a non-constant method and a constant method, an example using a *const-cast*, as well as an example of using a nested `State` type that is passed a mutable reference.

To facilitate thread-safety, ACTS is written to be *const-correct*. This largely means that the `const` keyword in C++ is used extensively wherever mutability is not required. Additionally, constructs which circumvent the `const` keyword are not allowed. The `mutable` keyword and `const_cast<T*>(const T*)` are examples of such disallowed constructs. Code without mutable state is frequently thread-safe automatically, as read-only access to data does not require synchronization.

Four examples of methods which use mutable state are shown in listing 6.4. It shows a `struct Tool` that contains member variables and methods using the variables. The first

method is not marked `const`, and is thus free to modify its member variable `m_value`. This fact might not be obvious to the caller, the method cannot be called at all if the caller only has a constant reference or pointer. The second method uses the `mutable` keyword to have the compiler allow it to modify `m_value2`, even though the method is marked `const`. The third method circumvents the fact that the `this` pointer is `const` due to the method being marked `const`, by using a const-cast. This enables it to write to `m_value3`. These two examples are particularly devious, since the `const` keyword suggests to the client that no internal state is mutated. They might therefore use these methods in a way that breaks thread-safety. Finally, the fourth method uses the encouraged pattern in ACTS: a dedicated `struct State` holds mutable state. An instance of this type is passed explicitly as a mutable reference to the `const` method. It is assumed to be held in a thread-local way by the client, the method is free to mutate it. If the client complies, no synchronization is needed to achieve thread-safety.

In many reconstruction scenarios, information that changes between events is required. This type of data is often referred to as *conditions* and can range from varying voltage values in hardware components, to the alignment of detector components. This information needs to be provided to the routines that need them. This is easily implemented in sequential processing, but becomes more involved once multiple events can be processed at the same time. Here, multiple sets of conditions data must be made available to clients depending on which event they are operating on. Since ACTS strives to be thread-safe, this case must be handled. The solution is the extensive use of explicit context objects, that are valid for a given IOV. Depending on the specific implementation, retrieving the current context and the corresponding information can be an expensive operation. This can be mitigated by performing the retrieval of information as far up in the call chain as possible, for example at the event level. Retrieved contexts can then be passed down the call chain to routines which need them without incurring further retrieval costs. Concurrent handling of conditions information on the example of sensor alignment will be further discussed in section 7.7. The same approach is also used to deal with changing magnetic field data and measurement calibration.

Although ACTS does not have a dependency on a specific concurrency backend, set of examples is provided that uses Intel's Threading Building Blocks (TBB) [186] library. These examples are used to test thread-safety, and ensure numerical results are identical between serial and concurrent execution.

6.3. Components

The ACTS package contains a number of components that integrate with each other to provide various functionality. These components can be further differentiated into *core components* that are always part of an installation, and *optional components*, that can individually be enabled or disabled. Figure 6.2 shows an overview of the core components, which are always available. Arrows between some of the components indicate the interplay between the components non-exhaustively. Examples of this are the navigator using the geometry description,

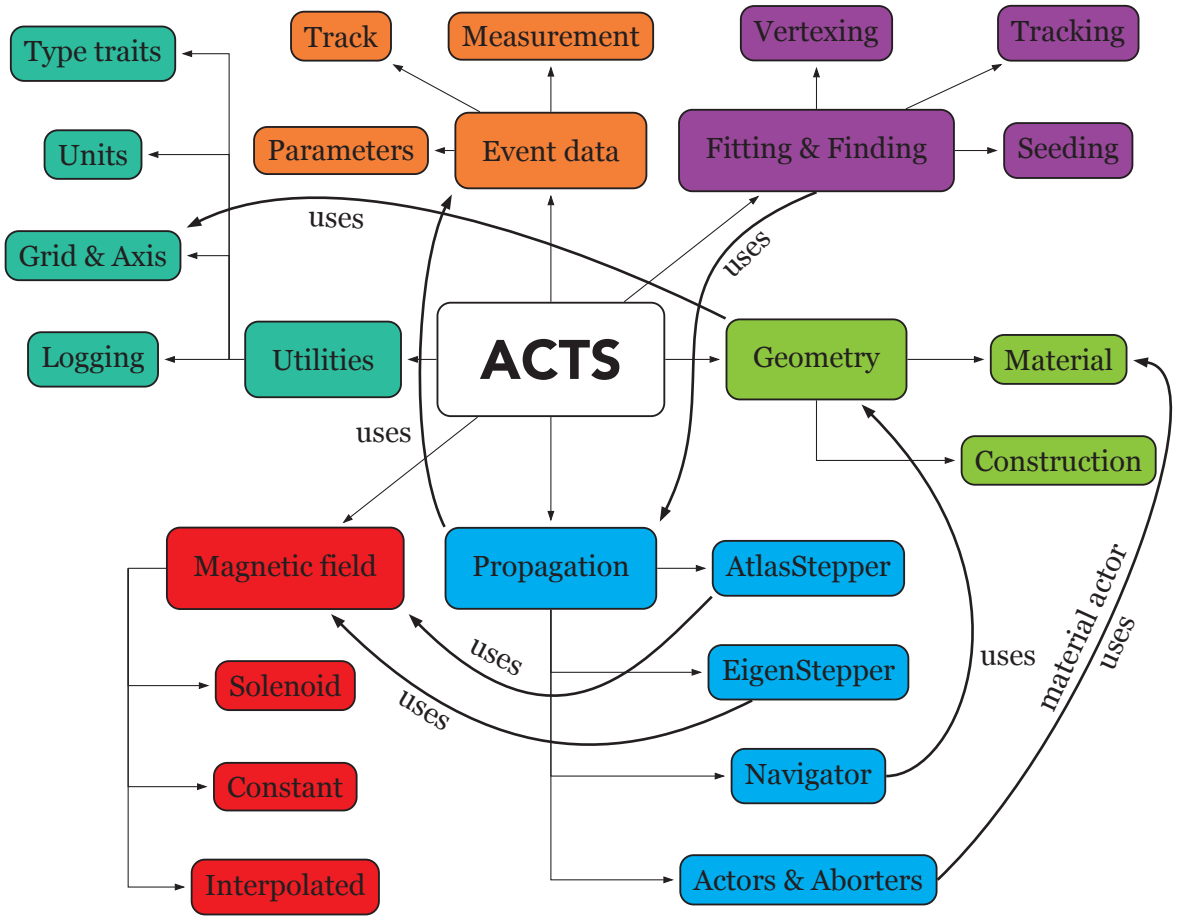


Figure 6.2.: Overview of the various components that are part of ACTS. It shows larger components like Geometry, Propagation, Magnetic field, Fitting & Finding, Event data and the Utilities. Also shown are non-exhaustive lists of sub-components, as are some connections where components use each other.

the steppers querying the magnetic field, and the fitting using the propagation. These are needed in the context of most HEP experiments.

An exhaustive list of optional components available at the time of writing can be found in table 6.1.

The core components are designed to have only very limited required dependencies. Apart from the two mandatory dependencies `Eigen` and `Boost`, other dependencies are only enabled if the corresponding optional component is enabled as well. By providing interfaces in the core modules, the optional components, or *plugins*, can provide additional functionality and compatibility with popular external software.

The following describes some of the core components in more detail, and outline specific decisions that were made in their development.

name	purpose
TGeo	Plugin to build detector geometries using the TGeo description from the `ROOT` [187] software. It therefore adds a dependency to `ROOT`. Code to automatically detect layouts exists, but manual connecting code might still be required.
`DD4hep`	Plugin to build detector geometries using descriptions from the `DD4hep` [188] format. Some additional code might be required, depending on the exact implementation of the `DD4hep` geometry in question.
JSON	Enables JSON reading and writing using the `nlohmann::json` [189] library for C++. Currently this includes a reader and writer for geometry material description.
Digitization	Plugin which can perform digitization of particle charge deposits. This plugin is not designed to be exhaustive, and does not take into account the potentially sophisticated details required for various experiments.
Identification	Plugin that provides a minimal identification implementation. This is required for some examples, but experiments are encouraged to use their specific identifier schemes, which ACTS is fully compatible with.
FATRAS	Fast track simulation package which uses the ACTS propagation package and integrates energy loss and scattering effects.
Examples	A number of examples using an internally developed testing framework. It can be executed concurrently by leveraging the `TBB` library. Examples for simulation, digitization, propagation, geometry construction, track finding and fitting, vertexing are provided among others.

Table 6.1.: Table of all available optional components to ACTS at the time of writing. Each of them can be individually turned on and off at build time. Continuous integration tests ensure that the code base can build with and without the various plugins enabled.

6.3.1. Geometry description

Geometry information is a critical input to many track reconstruction algorithms. The geometry modelling component in ACTS closely resembles the one outlined in section 4.4, as it is based on the ATLAS implementation. It constructs a simplified version of a more detailed geometry, which is used for simulation, by only explicitly modelling sensitive surfaces. Passive surfaces are replaced by an empirical approximation of the distribution of their material through a mapping procedure. Here, the passive material seen by special probe particle trajectories is projected onto carefully selected representative surfaces. When a particle trajectory passes through such a material surface, it can retrieve the effective amount of passive material that the numerical integration should account for. As generally only averaged material properties are needed for particle propagation, this modelling of material is often sufficient. Even though the geometry description is similar to the one found in ATLAS, there are a number of differences. The ACTS implementation removes features that are not relevant in an experiment independent context. An example is removal of the explicit knowledge about the identifier scheme. This knowledge is restricted to the associated detector element object, which is experiment specific regardless. Another difference is the concept of ownership, which

is removed in favor of an automated solution (see section 7.6). Finally, direct coupling to the details of the track parametrization is removed.

6.3.2. Event Data Model

An Event Data Model (EDM) enables storage of required input data for reconstruction algorithms, as well as their outputs. The ACTS EDM aims to strike a balance between being lightweight and powerful. Ideally, conversions between any experiment specific EDM and the ACTS specific one are kept to an absolute minimum, since every conversion incurs a computational cost. The basic underlying idea is an enumeration of the parameter indices of the local track parametrization. This enumeration can be supplied by client code, but ACTS includes a default parameter set assuming two-dimensional sensors:

$$\vec{x} = (l_0, l_1, \phi, \theta, q/p, t)^T . \tag{6.2}$$

Here, $l_{0,1}$ are the local spatial coordinates, $\phi \in [-\pi, \pi)$ is the azimuth and $\theta \in [0, \pi]$ the polar angle (see figure 4.2). q/p is the fraction of charge over momentum. In case of neutral particles, this parameter is interpreted as $1/p$. Finally, t denotes the time. Aside from the time parameter, this parametrization coincides with the one used in ATLAS. In order to compare track parameters at the intersection of a trajectory and a sensor to a measurement of that sensor, the global track parameters need to be converted to this local representation of the sensor, and then converted to the measurement frame.

In the ATLAS software it is possible for the measurement frame to be different from the local coordinate system of the sensor. An illustration of this can be found in figure 6.3 for the example of a silicon strip sensor in the shape of an annulus section. Natively, in a sensor consisting of a single row of strips, the local measurement frame is one-dimensional and aligns with the strip pitch direction.

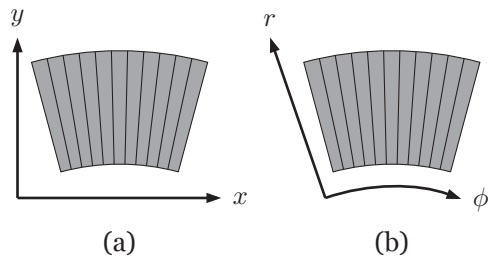


Figure 6.3.: Illustration showing the different coordinate frames for the example of a strip endcap module. Shown are a cartesian frame (a), and the corresponding local polar coordinate frame (b) as seen from the global frame.

In case of an endcap module, this strip pitch direction is the ϕ unit vector, since the strips all point toward the theoretical disk center. Figure 6.3a shows a mismatch between coordinate systems, where the local-frame is defined with cartesian coordinates. The part of the ATLAS

tracking library dealing with the endcaps of the SCT uses a similar approximation. It does not match the native measurement frame of this kind of strip module, which is shown in figure 6.3b.

In contrast to the cartesian approximation, the ACTS EDM makes the rigorous assumption that the measurement frame can only be a strict subset of the parameters from equation (6.2). As discussed in section 4.7, in the context of a Kalman Filter [160], the relationship between a measurement $\vec{m}_k$ for state k, and the track parameter vector in the same state is

$$\vec{m}_k = \mathbf{H}_k \vec{x}_k + \epsilon_k. \tag{6.3}$$

where ϵ_k is the measurement noise. It also features the measurement mapping function $\mathbf{H}_k$, which is required to be linear. Subsequently, it is used for the filtering of the parameter vector using

$$\vec{x}_k = \vec{x}_k^{k-1} + \mathbf{K}_k \left(\vec{m}_k - \mathbf{H}_k \vec{x}_k^{k-1} \right), \tag{6.4}$$

among others. Here, $\vec{x}_k^{k-1}$ is the prediction of the parameter vector at state k based on the previous state $k-1$. $\mathbf{K}_k$ is the Kalman gain matrix. With the assumption detailed before, $\mathbf{H}_k$ becomes a simple projection. For the case of a measurement of the l_0 and l_1 parameters, it then looks like

$$\mathbf{H}_k = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}, \tag{6.5}$$

A concrete example where the difference between the local sensor frame and the measurement frame becomes important are discussed in section 7.3.3.

Technically, the implementation of the ACTS EDM consists of a measurement class:

```
template <typename source_link_t, ParID_t...> class Measurement;
```

The first template parameter is a type fulfilling the `SourceLink` concept, and is used to associate the measurement with a potentially experiment specific data source. For the simplest case, this is a pointer or an index into a collection of raw inputs. In track reconstruction, the properties of a measurement can often only be interpreted correctly in the context of a track hypothesis. Using information of said hypothesis, measurements can be calibrated geometrically, for example for treating wire sagging, or deformed modules. In the processing chain used in ATLAS, and largely mirrored in ACTS, raw measurements are calibrated in this way before comparison with intersections of a track trajectory and the respective sensor. To minimize conversions, ACTS consumes the raw inputs identified by a source link, using a provided experiment specific calibrator. Since this step is necessary even if the implementation was completely aware of the experiment, conversion overhead should be kept to a minimum. `ParID_t...` is a variable number of parameter indices, where the dimension of the measurement is equal to the number of supplied indices. From these indices, which

are known at compile time, the class can generate highly optimized code for mathematical operations, and provides a projection matrix like the one shown in equation (6.5).

One complication is the storage of these measurements in a homogeneous way, often called *type-erasure*. In ACTS this is achieved using the `std::variant` type from the C++ Standard Template Library (STL). It provides a way to define a container that is able to store any concrete type out of an explicitly provided list, but presents as a single homogeneous type. It also stores information on which type is contained. Using another STL member, `std::visit`, it is possible to dispatch code on the concrete stored type, thereby enabling optimized code paths. The remaining task is to generate an `std::variant` with all possible measurement, which is discussed in a bit more detail in section 7.5.1. Another aspect is that this method of type-erasure has certain drawbacks depending on the context. Another approach working hand-in-hand with the implementation outlined above is shown in section 7.5.2.

6.3.3. Particle propagation

The properties of a particle track at a point further along its path can be estimated based on the properties at the current location. This process is called particle propagation and is central to track reconstruction (see section 4.3). The ACTS implementation of particle propagation makes some changes with respect to the ATLAS propagation. The most notable change, as already described in section 6.1, is the consistent switch to `Eigen` for linear algebra operations. Figure 6.4 shows an overview of the architecture of the propagator in ACTS. The ACTS propagator is fundamentally designed to be extensible. Code dealing with the numerical integration, if required, is separated from the code steering the propagation itself. While the general approach was already present in ATLAS, the implementation there involved using virtual inheritance to separate the functionality. This implies very frequent virtual table lookups during propagation. ACTS' propagator uses templating to be aware of the concrete type of the numerical integrator, named the *stepper*. By doing this, it allows forgoing the virtual function calls and directly calling into the various required methods. The propagator accepts a *stepper* and a *navigator* as mandatory components. The stepper handles numerical integration, and is also aware of the magnetic field used in the propagation. On the other hand, the navigator takes care of figuring out which parts of the geometry the particle trajectory will encounter, and will have to be handled by the propagation.

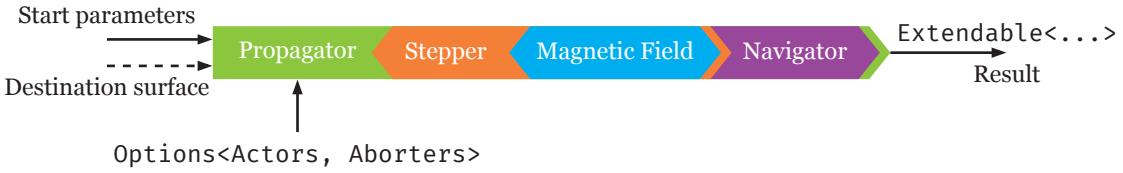


Figure 6.4.: Diagram showing the architecture of the ACTS propagator. It shows the inputs, starting parameters and an optional destination surface, and the extendable output. The functionality is controlled by the client provided options containing actors and aborters, and a number of mandatory components.

Aside from these mandatory components, the propagator can be configured further with optional components. During setup, two lists of *actors* and *aborters* are supplied to the propagator. Each actor defines a result type, that it wishes to populate. The aborters can specify a dependency on any of the actors' result types. During propagation, the propagator will loop over the list of actors at every step and invoke them with the current mutable state of the propagation and the stepper. The lists are known at compile-time, so the loop requires little to no indirection. Actors are free to modify the mutable state, as well as accumulate data in their respective results. Before every step, the propagator will then consult the list of aborters whether the propagation should be terminated. Each aborter is supplied with the propagator and stepper state as well, complemented by the result the aborter specifies. At the end of a successful propagation, all actor results are returned by the propagator invocation, and can be consumed by client code. This actor/aborter model can be used to configure the propagator in a wide variety of ways. Features such as on-the-fly search for missing expected hits, or a path limit that aborts the propagation can be granularly added. Both the ACTS Kalman Filter and the Combinatorial Kalman Filter are implemented using actors, and thus run during propagation. This has the benefit of avoiding unnecessary navigation operations, or coordinate transformations, as the ones used for the propagation itself can be reused.

The main implementation of the numerical integration, the *EigenStepper*, is customizable even further. At the lowest level, it is possible to modify the details of the numerical integration, which is necessary for example to incorporate material effects in a continuous way, rather than the discrete method described in section 4.3.

6.3.4. Magnetic field lookup

Momentum measurement by means of reconstructing the curvature of particle tracks requires the presence of a magnetic field to cause bending. Applications such as the particle propagation therefore require detailed knowledge of the magnetic field. ACTS' magnetic field infrastructure is designed with performance and adaptability in mind. The public interface abstracts away the concrete source of magnetic field data. Included is an implementation that can interpolate a magnetic field from an equidistant grid of field values, optionally factoring in symmetries in order to operate on sparser input data. Tools are provided to read magnetic field data from `ROOT` and Comma-separated-values (CSV) inputs. An analytically calculated solenoid field provider is also included, that can be configured with a number of parameters.

One approach to provide magnetic field data is to interpolate between a grid of known field vectors. This is implemented in ACTS in the `InterpolatedBFieldMap` class. The class can calculate the magnetic field at any given point inside the grid by retrieving the surrounding grid points, and interpolating between them. The set of surrounding grid points is referred to as a *field cell*. It is possible to use symmetries, for example a radial symmetry around the z-axis, to reduce the size of the required grid.

To optimize field access performance, magnetic field providers are able to use caching. Depending on the concrete type of the field source, the way a cache works can vary. For the case of the interpolated magnetic field, the cache contains a field cell. As illustrated in

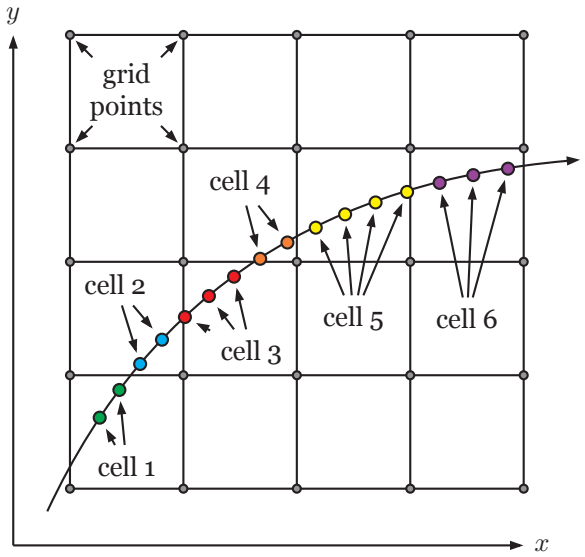


Figure 6.5.: Illustration of the magnetic field cell implementation. This example is for a two-dimensional field map in the xy-plane. The colored circles represent propagation steps where a magnetic field lookup is performed. Before crossing the boundary into the next cell, each step can reuse the previously retrieved field cell.

figure 6.5, the idea is based on the circumstance that consecutive field requests will very often only move a small amount and end up in the same grid cell as the previous call. The figure shows consecutive positions, at which the magnetic field needs to be retrieved by the numerical integration. For positions of the same color, the field cell retrieval will return the same call as previously. By caching this field cell and only retrieving a new one if the current position is outside of it, field access performance can be significantly improved.

```
struct FieldProvider {
  struct Cache {};
  // get the field for a given position, and provide the cache object
  Vector3D getField(const Vector3D& position, Cache& cache) const;
  // ...
};

// client code
FieldProvider p;
FieldProvider::Cache cache;
auto field = p.getField({1, 2, 3}, cache); // retrieve field
```

Listing 6.5.: Example of a field provider which exposes a `Cache` type. The implementation of the cache should be opaque to the client. The provider requires the client to provide an instance of the cache object that is thread-local and safe to write to as a mutable reference argument for a field request.

Other magnetic field providers are free to implement a suitable method of caching. However, the providers need to conform to a certain interface, and need to expose a `Cache` type. An example for a pseudo field provider is listed in listing 6.5. The `struct FieldProvider`

implements the `getField` method which accepts a position, and an instance of the nested `struct Cache`. The client is supposed to instantiate this cache type and provide the instance to every field request. The cache is assumed to be thread-local by the field provider. In the case of the interpolated field provider, the generic cache is used to store the field cell that was discussed above.

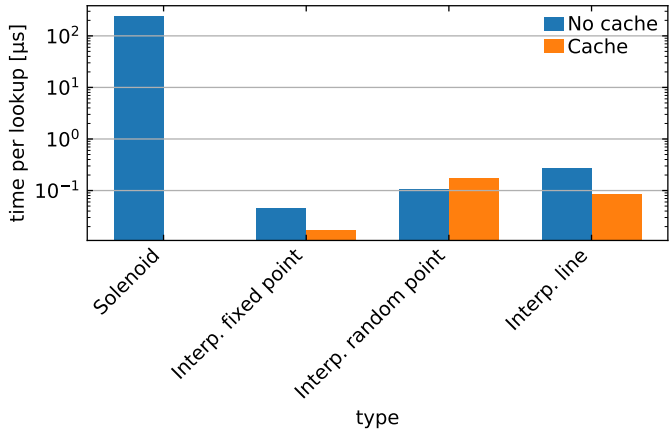


Figure 6.6.: Performance comparison for different field providers and access patterns. Shown are an analytically calculated solenoid field, and interpolated lookups for fixed, random, and subsequent positions along a line, with and without caching.

The impact of this caching mechanism can be seen in figure 6.6. It shows timing measurements for the analytical solenoid field provider mentioned before, as well as the interpolated version of the same field with different access patterns. The interpolation grid is produced directly from the analytical solenoid field before the timing measurements are started. The access patterns being compared are repeated lookup of the same position, lookup of random positions and lookup of positions with equal steps along a fixed direction. Retrieval of field values at fixed and random locations is not a typical access pattern in track reconstruction, but is useful for a performance benchmark. The straight-line access pattern models the propagation of a high-momentum particle, whose trajectory has minimal curvature.

The direct access to the analytical solenoid is clearly slowest by two orders of magnitudes. For repeated access of the same fixed point the time per lookup is below 100 ns, in this case the cache improves the performance by reasonable amount. In contrast to this, in case random points are accessed, the performance actually decreases when using the cache. This can be explained by the fact that to check whether the cache needs to be refreshed, a test to see if the lookup position is in the current field cell needs to be executed. This test is most likely going to fail every single time for the random access pattern, meaning that in almost all cases, the actual lookup will have to be performed in addition to the cache check. For the access pattern that emulates the linear particle extrapolation, the cache improves performance again. In this case, the cache is still valid a lot of time, so the full field lookup can be avoided frequently. The difference in performance between the three interpolation scenarios without explicit cache can be explained with the additional intrinsic memory cache facility of the CPU. The

processor keeps recently accessed locations of memory in a very fast cache. For a fixed point, this intrinsic cache is very efficient. In the other scenarios it is not, because the CPU can not predict correctly what should be cached. This caching happens automatically, and is in addition to the explicit caching mechanism employed in ACTS.

6.4. Software development best practices and implementation

From a more technical point of view, ACTS tries to achieve maintainability and high code quality by adopting a number of workflows and best-practices that are widespread in the software industry.

6.4.1. Build system and version control

Version control of source code allows tracking changes that are made over time. A build system, in the context of compiled programming languages, orchestrates the compilation process.

At the time of extraction, the ATLAS software was built using the Configuration Management Tool (CMT) [190] build system, and was version controlled using Subversion (SVN) [191]. Due to CMT usage at large being relatively low, as well as Version Control Systems (VCSs) like Git [192] becoming more and more popular, ATLAS was looking to migrate the software to more modern and industry standard solutions. ACTS was used to prototype usage of Git, as well as the intended replacement for CMT, the CMake [193] build system.

CMake is used in ACTS to enable great flexibility to configure and adapt to the requirements of the project. In the context of ACTS this manifests itself in the use of CMake configuration options to steer the build of the core software, as well as the optional plugins mentioned above. A full list of all available options can be found in table A.1.

Git is a *distributed* VCS. In contrast to prior VCSs, every working copy of a `Git` repository is a full copy. Operations such as committing changes, and even manipulating branches and tags can be performed locally, without the need to communicate with a central server. Apart from that, Git also made branches a primary concept, rather than just being an afterthought, as was often the case before. *Merging* changes from one branch to another is a safe, mostly automated operation, that has become a basic building block of collaboration models.

Next to the lower-level version control system Git, a range of collaboration platforms have developed on top of it. Solutions like GitHub [194], GitLab [195] or Gitea [196] enable web-based interaction between collaborators. The main mode of collaboration is the *pull-* or *merge-request*. These constitute an interface helping with building, discussing and eventually accepting code changes on top of the lower-level merge process. Usually, this is done across different copies of the source repository (*forks*). Effectively, this enables anyone to contribute to any public project on these platforms, and has been become extremely popular in the open-source world. Thus, ACTS uses the same model. The project was initially hosted on CERN's own instance of GitLab [197] but moved to GitHub [198] in March 2020. Pull requests

are subjected to an informal review process by one of the core developers. The review covers general comments on code style, compliance with the overall design goals, and detailed discussions on aspects on the contribution. The review process is augmented by automated checks (see section 6.4.3).

6.4.2. Unit and integration testing

Unit testing describes the practice of writing and maintaining automated tests that verify the behavior and results of a *unit* of code. The exact definition of what constitutes a unit varies, ranging from individual functions to larger constructs. In its simplest form, a unit tests invokes a portion of code with known input, and asserts whether the results match the expectation. Depending on the design and structure of the code, this can be more or less difficult. For instance, code that causes or depends on side-effects, essentially everything that is not an explicit input or output, is typically much harder to test. Tools to *mock* other components that a portion of code uses, or setting up known environments (*fixtures*) can be leveraged to make isolated testing more manageable.

By decoupling units and testing them individually, tests can usually be reduced in size, and become more granular. The lower the amount of (potentially hidden) code a unit test covers, the easier it is to identify failures and thus problems, and trace them back to code changes that were made. Another closely related approach is *test-driven-development*. Here, the test is written before the implementation itself. The intended behavior is encoded in the test, which can help with formulating the requirements the portion of code is supposed to fulfill.

In some cases, the complexity of testing can become large. This can be the case when code would have to be fundamentally refactored, in order to be able to test its parts individually. Additionally, sometimes verifying the result of a chain of operations serves as a more reliable test of correct behavior than individual pieces. In these cases, higher-level *integration tests* can be used. Lacking again a strict formal definition, these are usually interpreted to touch on a larger number of components or units, whose results are then verified against an expectation in combination. While granularity can make it more difficult to trace down the source of failures, these tests can also more faithfully model real-world scenarios. One example of an integration test in ACTS is the stepper comparison. Here, different implementations of the particle propagation algorithms are compared with one another, to ensure they yield consistent results.

Unit testing in ACTS is a core aspect of the development workflow. New code is expected to include tests, a requirement that is enforced in a code review conducted by the core contributor team. Integration tests are added for parts of the code where strict unit testing has proven to be cumbersome. In addition, they are used as end-to-end tests for groups of functionality, such as the numerical integration, or trajectory fitting.

The extent to which unit testing is applied to a code base is often expressed in terms of *coverage* of the code. This quantity is defined as the fraction of code that is exercised when automated tests are run. There are different approaches to calculate code coverage. The simplest approach is line coverage, where the number of executed lines are divided by the

total number of lines in the code. Another approach is to measure the fraction of branches. Branches can range from constructs as simple as

```
if(some_condition) {/* branch A */} else {/* branch B*/}
```

to more complicated scenarios. While branch coverage is more difficult to extract, it gives a more realistic insight into how well the code is tested. Note that in both cases, coverage only measures *that* code is executed from tests. It does not check the code for correctness. Code coverage should therefore not be used as the single metric to evaluate the extent of unit testing. At the time of writing, the test suite in ACTS covers around 45 % of branches in the code base.

6.4.3. Continuous integration

Besides running tests manually during development, there is benefit in automatically testing the code. This is commonly referred to as *continuous integration*. In ACTS it is implemented by leveraging GitHub's *Actions* feature, which allows defining pipelines that run on versions of the code. Pipelines are used to check success of all included unit and integration tests, as well as conformance with the format guide and correct license statements. These pipelines are triggered when receiving new commits, as well as in the context of pull-requests. In a pull-request, the successful completion of these pipelines is a precondition for accepting it. The code is compiled on a number of platforms, ranging from a standard Ubuntu installation, versions of Scientific Linux CERN 6 (SLC6) [199] and CERN CentOS 7 (CC7) [200] using the LHC Computing Grid (LCG) [201] software stack, to a macOS build. Unit and integration tests are required to pass on each of these platforms. Automated compilation such as this can ensure that changes made by a developer on a specific platform do not unintentionally break the software on another one.

6.5. Recent developments

Development of ACTS is ongoing. Aside from the initial contributors, which are mainly involved with the ATLAS experiment, there are a number of contributors from experiments such as Belle-2 [202], the CEPC [203] detector, FASER [204] and sPHENIX [205] evaluating the software for deployment in their respective track reconstruction applications. This means the software is rapidly evolving. In the following, some recent developments, at the time of writing, are discussed, that are not directly associated with work for this thesis.

Based on the storage model that is the topic oc section 7.5.2, ACTS recently merged the implementation of a CKF. The implementation is closely modelled after the (single) Kalman Filter [160], that has been part of the code base for a longer time. It is implemented using an actor in the propagation, and consumes source links that are indexed by measurement surface. It can automatically determine missing measurements on intersected sensitive surfaces while traversing through the geometry.

The propagation component was augmented with the ability to handle time information during numerical integration. This allows calculating the time-of-flight of the particle at every integration step. Using this method, detectors which measure the time of particle intersection can be used to update the trajectory. Since the measurement type in the ACTS EDM (section 6.3.2) can easily be extended with additional parameters, and the projection mechanism works identically, an extra time parameter is natively supported. Using the extension mechanism of the `EigenStepper`, the propagation was supplemented with an implementation of the continuous treatment of energy loss and multiple scattering, dubbed Simultaneous Track and Error Propagation (STEP), as described in [152].

Finally, implementations of multiple algorithms for vertex finding and vertex fitting have been added based on ATLAS versions of them. This includes an iterative and an adaptive multi-vertex finder [165] as well as fitters based on the Billoir algorithm [164] and the adaptive multi-vertex approach.

7. Development and improvement of the ACTS software

This chapter describes the work on ACTS carried out in the context of this thesis in more detail, in contrast to chapter 6, which provided a more generic view of the project. Work has been done in a number of sectors that are essential in moving ACTS toward a production-ready toolbox, as well as redeployment as part of the ATLAS software. The chapter therefore contains a description of the integration of ACTS into the Athena software framework in section 7.1, followed by a discussion of the geometry modelling of the current ATLAS *Inner Detector* in section 7.2. It is accompanied by the description of its eventual replacement, the ITk (see section 5.3.1), in section 7.3, and in particular the handling of strip modules in the endcaps. The implementation of a new navigation model for arbitrary geometries is shown in section 7.4, alongside results from the application of said model to the ATLAS calorimeter geometry. Another topic is the evolution of the ACTS EDM, with particular focus on its compile-time aspects in section 7.5. There is also a report on the implementation of a column-base storage model for track states, that is used for the ACTS Kalman Filter, and is specifically designed for the recently added CKF. Finally, section 7.7 touches on the design and implementation of a system for efficient and correct handling of concurrent alignment of sensitive elements. This is complemented with information on the implementation in the ATLAS software, that integrates the concurrent conditions facilities with the ACTS implementation.

7.1. Integration of ACTS into the ATLAS software framework

As was discussed in section 6.1, while many ACTS components are extracted or derived from the ATLAS software, an eventual deployment of components back into that framework is envisioned. Various steps have to be taken in order to leverage ACTS tools in this environment and were carried out in the context of this thesis. This chapter reports on how the integration is achieved, and on aspects of the modelling of the ATLAS geometry using ACTS.

7.1.1. ATLAS environment

The first basic step is making the code available from inside the ATLAS Athena [109] environment. To understand the deployment method, it is useful to introduce the way the software is configured and built. The Athena software is built using the same CMake build system as ACTS, albeit with a large amount of advanced configuration to accommodate ATLAS requirements. This is not a coincidence, as ACTS was used as a test environment for the subsequent

migration of the ATLAS software from CMT to CMake. The ATLAS CMake infrastructure contains a large amount of sophisticated custom code that enables a build model that makes sense for the project. In the previous CMT-based build infrastructure the build was centered around a large number of *packages*. Each package is confined to a single folder, and consists of C++ header and implementation files, as well as auxiliary files, such as the job configuration in Python. CMT enabled declaring dependencies between individual packages, which was transformed into a dependency tree and used to orchestrate the build. Each package was individually managed in source control, first in Concurrent Versions System (CVS), and subsequently in SVN. Packages followed individual version strategies, leading to highly complex version requirement structures, that made updating packages very difficult.

Due to the size of the codebase, about 9 million lines of mostly C++ and Python code, building it takes considerable amount of time. To enable reasonable build times for individual developers, a nightly build of the entire project is conducted. Using this nightly, developers can check out a local copy of the source code, activate a certain nightly build, and only build selected packages against this build. As a project maintained by a large collaboration of developers, this model helps keep the development reasonably easy.

During the transition to CMake, this general usage pattern needed to be supported, even though the concept of a package does not natively exist in CMake. The way CMake handles dependencies is by defining a series of *targets*, which can be executables, libraries and a few others. CMake allows *importing* targets, which can be used to model the case where such targets are compiled separately. To accommodate the workflow of building a few packages against a complete shared build was introduced in ATLAS specific CMake configuration code. The technical concept of a package was dropped, dependencies are modelled using CMake targets. The logical model of a package, however, was kept for organizational purposes. To address the substantial work required to compile a list of mutually compatible package versions, all packages were merged into a single repository in Git. With a *monorepo* structure like this, all changes need to fulfill the requirement that the full projects still builds. This is enforced using continuous integration in addition to the nightly builds.

Aside from the packages that make up the majority of the software, the ATLAS software also depends on a series of external projects that need to be part of the build process. Critically, this is the Gaudi [110] framework, which provides the event loop and a number of other pieces of infrastructure used for processing. Other examples are linear algebra libraries such as `Eigen`, `CLHEP` [180] or `LAPACK` [206], the simulation framework `Geant4` [132], TBB [186] or `ROOT` [187]. These external dependencies are built from a separate repository before the main build, such that all packages can leverage them.

7.1.2. ACTS specific packages

Since May 2018, ACTS is built as part of the ATLAS *Externals* project. Initially, the build tracked the *master* development branch of ACTS, but was since changed to an explicit version to prevent unexpected build problems due to code changes in ACTS. While this makes the shared libraries and headers available from ATLAS packages, code that uses ACTS still had to

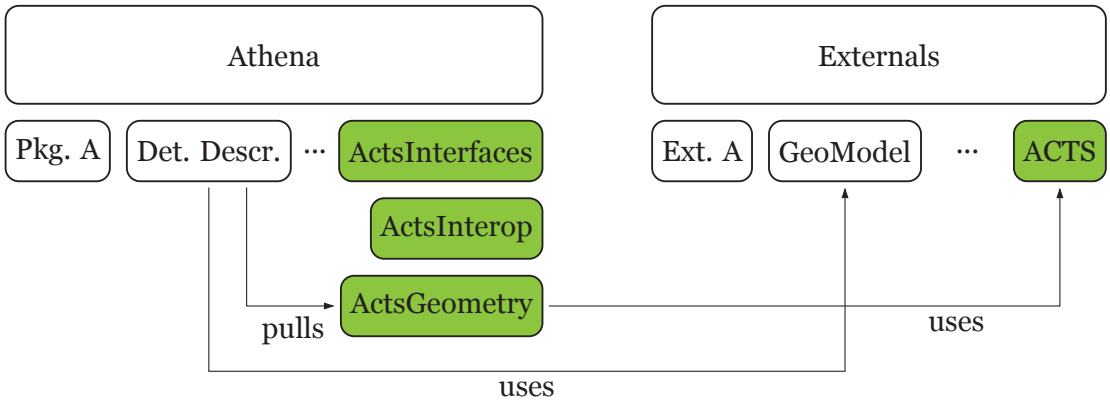


Figure 7.1.: Overview of the build structure of ACTS in the ATLAS software. The *Athena* and *Externals* projects are shown, alongside some packages contained in them. A number of ACTS aware packages are shown under Athena, and the ACTS package is located within the Externals.

be written.

Figure 7.1 shows an overview of the Athena and Externals projects. Both consist of a number of packages, with some examples shown in the figure. This notably includes an ACTS package under the Externals, that builds the library. The Externals also contain the `GeoModel` library, that was extracted to an external package from one contained in Athena. This package is used by a series of packages grouped as *Detector Description* to provide a detailed description of the geometry of the entire ATLAS detector.

Information from `GeoModel` and the *Detector Description* packages are pulled into a package called `ActsGeometry`, that uses it to construct ACTS specific instances of the various geometry classes. The collection of these resulting classes make up the tracking geometry that can be used by other ACTS tools and algorithms also provided by the package for particle propagation and debugging. Details on the geometry construction procedures are the focus of the following sections 7.2 and 7.3. Since the example algorithms and tools are mostly designed around the concurrent alignment infrastructure that is described in section 7.7, they are also discussed in that section. The package also contains a magnetic field provider complying with the interface introduced in section 6.3.4 that uses the ATLAS magnetic field service as the data source. A package called `ActsGeometryInterfaces` is provided that defines the Application Programming Interface (API) of the aforementioned components and tools.

Apart from the geometry related aspects, an additional `ActsInterop` package enables a unified log handling that leverages the Athena logging facilities. The ACTS logging solution allows fine-grained configuration and customization. In particular, the backend used for displaying log messages can be switched out. With the aforementioned package, each ACTS logger instance can be configured to pass log messages to an Athena logging component, thereby using the settings taken from the Athena Python configuration.

7.2. Description of current ATLAS geometry

This section describes the details of how an ACTS representation of the ID geometry is constructed. The geometry forms the basis for a significant portion of tasks related to track reconstruction. As described in section 4.4, a model of the geometry is used which is simplified relative to the more detailed simulation geometry. This stems from the realization that while highly detailed information is necessary to model small effects that are relevant in simulation, during reconstruction, it is usually safe to approximate these effects, as only averaged material properties enter most calculations.

7.2.1. GeoModel geometry

In ATLAS, the fundamental source of geometry information is implemented across various packages using the `GeoModel` [207]. The `GeoModel` description works with a tree of volume nodes, each of which stores a transformation matrix relative to its parent. To obtain the absolute transform in the global frame of such a node, the tree needs to be traversed upwards until the root volume is reached, and the transformation matrices need to be accumulated. The top volume corresponds to the global frame. `GeoModel` implements mechanisms to parametrize these transforms. This is useful in cases where, for example, an element is repeated multiple times, each instance having a slightly modified rotation.

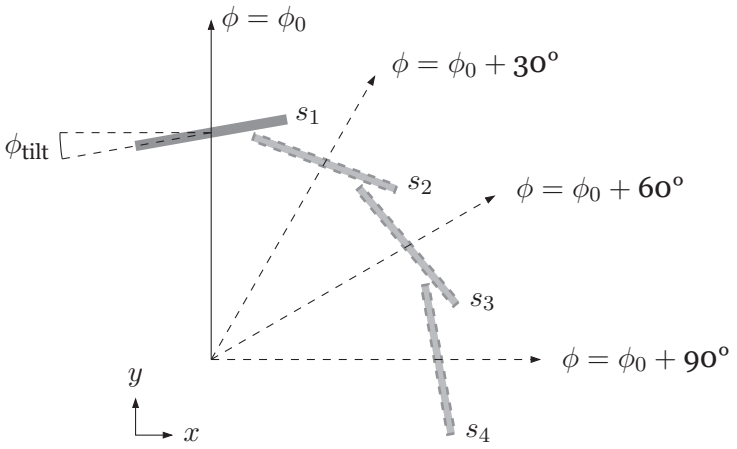


Figure 7.2.: Illustration of a parametrized transform being used to described multiple instances of equivalent geometry objects. Shown is one *real* sensor instance at $\phi = \phi_0$, and three rotated copies with offsets in ϕ. Also shown is a tilt angle of the sensors.

Suppose the geometry consists of a series of sensors rotated around the origin in the xy-plane, as seen in figure 7.2. One prototype sensor is defined at $\phi = 0$. Other equivalent instances, which only differ in their rotation angle ϕ are shown in dashed lines. The full transform T_{tot} can be described by the product of three affine transformation matrices

$$T_{\text{tot}}(\phi, s, \phi_{\text{tilt}}) = T_{\text{rot}}(\phi) \times T_{\text{shift}}(s) \times T_{\text{tilt}}(\phi_{\text{tilt}}). \tag{7.1}$$

The entire sequence of elements can be described by modelling the product in equation (7.1) in `GeoModel`. This way, rather than precomputing the rotation matrices, and describing essentially equivalent copies of the same nodes, the prototype, or *logical*, node can be dynamically duplicated. When traversing the tree, the parametrization is automatically resolved. While the example from above performs a simple rotation, the mechanism is generic, and can be used to almost arbitrarily parametrize geometric components.

Using this mechanism, the ATLAS ID consists of about 18 000 logical `GeoModel` volumes organized in such a hierarchy, which form about 1.8×10^6 total volumes when fully instantiated [208]. As a comparison: the MS and calorimeters feature around 9000 (1.4×10^6) and 36 000 (1.6×10^6) logical (total) volumes in their geometries. In combination, the ATLAS geometry consists of 64 000 logical and 4.8×10^6 total volumes, weighing in at just under 100 MB of memory required to store the full `GeoModel` hierarchy.

Transforms can also be marked as alignable, which means that they store an additional modification matrix in addition to their nominal transformation matrix. This setup allows the geometry tree to react to slight changes in the position of individual parts of the detector over time. It is clear that traversing the entire tree every time the absolute transform of a node is required, and also evaluating arbitrary expressions to resolve parametrized transforms like the one shown in equation (7.1), are not feasible in any performance critical code. To combat this, `GeoModel` implements a caching mechanism, which essentially stores the absolute transform in each node. In contrast, both the ATLAS and ACTS tracking geometries use absolute transforms in all elements, avoiding the need to traverse a hierarchy frequently.

Historically, the `GeoModel` transform cache was implemented in a private member variable that enabled every node of the geometry to also store its absolute transform. The cache is also sensitive to alignable transforms: Every time the alignment changes, the cache needs to be invalidated.

This becomes problematic when the geometry is used concurrently, which the ATLAS software is migrating to. Event ranges during which the alignment and other time-dependent properties are assumed to be constant are referred to as Intervals Of Validity (IOVs). If only a single event, or events from a single IOV of the alignment is processed at any given time, the cache can be invalidated at the end of an IOV. The cache will then have to be rebuilt with the updated alignment data in the following event. In case of concurrent processing, multiple events can be processed at the same time. As a consequence, multiple IOVs can be *in flight* simultaneously. Therefore, the cache invalidation must happen for each execution thread individually, to ensure consistency. How many IOVs are actually treated in parallel depends heavily on the execution environment. Datasets heavily filtered by physical event topology are likely to have events from many different IOVs in quick succession, which could exacerbate the issue. Concurrent event multiplicity has not been comprehensively determined, at the time of writing. More on this can be found in section 7.7.

Building the `GeoModel` geometry hierarchy in Athena is initiated by the `GeoModelSvc`. It is configured with a number of *detector factory tools*, one for each subdetector. These factory tools call recursively into factory classes that build up the geometry. These factories

correspond to physical assemblies in the construction, such as endcap wheels or ladders of modules. At each stage, newly created volumes are registered in the `GeoModel` tree. Finally, the construction arrives at the individual sensitive sensor elements relevant for track reconstruction. The modules are built as volumes shaped like the physical detector hardware. For track reconstruction, a surface-based representation is more suitable than a volume-based one. Additional domain specific information also needs to be pulled in. That step is handled in the *readout geometry*.

7.2.2. Readout geometry

In the ATLAS tracking software, another layer of geometry information is used that builds on top of the `GeoModel` description from before. This layer is referred to as the *readout geometry*, since it logically models the way the sensitive elements are read out. The code for this representation lives in the `InDetReadoutGeometry` package for the ID. The common base class for the elements in the ID is the `SiDetectorElement` for the Pixel and the SCT, and `TRT_BaseElement` for the TRT. Both of these classes inherit from a common `GeoModel` volume class, that is part of the `GeoModel` hierarchy. Instances of theses classes combine the geometry information from an associated `GeoModel` element with information on the *design* of the particular sensor in question. While `GeoModel` models all elements as volumes, the readout geometry models them as surfaces of small but non-zero thickness, and augmented with design parameters. These parameters include aspects such as the exact shape of the sensor in a surface representation. This entails planar surface bounds for the silicon based detectors, and straw surface bounds for the TRT. In the case of silicon detector elements, the design also stores information on the segmentation of the active material. Detailed knowledge of the segmentation is essential when turning raw data into spatial representations relative to the sensor during clustering (see section 4.5).

The readout geometry elements are constructed when a sensitive element is encountered in the `GeoModel` build procedure. The classes accept the `GeoModel` volume, an object detailing the design and a number that uniquely identifies the sensor on construction, and bind the three together. The created elements are stored in a flat collection that is indexed using the identification number from before. The identification number is a coded 64 bit integer. It is the concatenation of a number of bit ranges where each range is an integer on its own. The concept is demonstrated in table 7.1 for the example of such a 64 bit integer with a value range of $[0, 2^{64})$. The integer is shown in hexadecimal representation. In this form, one character corresponds to a 4 bit group with a range of $[0, 16)$. The 64 bit integer consists of 16 such groups. Three ranges are indicated as masks in the hexadecimal representation. The value range of these masks is listed as well. In essence the mask selects which bits are supposed to be interpreted to form the component number.

Which components exactly are encoded depends on the subdetector, helper classes like `PixelID`, `SCT_ID` and `TRT_ID` handle the interpretation and allow access. Each element also stores pointers to adjacent readout geometry elements, to allow navigation between them.

bit pattern [hex]	type	value
`0x0000 0000 0000 0000`	64 bit integer	0
`0xffff ffff ffff ffff`	64 bit integer	$2^{64} - 1$
`0xff00 0000 0000 0000`	8 bit mask	$[0, 2^{8})$
`0x00ff ffff 0000 0000`	24 bit mask	$[0, 2^{24})$
`0x0000 0000 ffff ffff`	32 bit mask	$[0, 2^{32})$

Table 7.1.: Table showing how an integer can be used to encode multiple components by defining bit ranges. Shown the minimum and maximum value of a 64 bit integer, next to their hexadecimal representation. Three examples of bit masks expressed in hexadecimal values are also shown. Their respective value ranges are listed as well.

7.2.3. Tracking geometry

To facilitate navigation and logical grouping, another hierarchy is constructed from the readout geometry, which is specifically tailored to access and logical patterns in track reconstruction. In the ATLAS track reconstruction library, this is done by iterating over the collection of elements and sorting them into convenient logical structures. For the silicon based detectors Pixel and SCT, this logical structure is the *layer*. Referring back to the discussion of the simplified tracking geometry from section 4.4, a layer is a two-dimensional surface which groups together a number of silicon sensors. The underlying assumption is that a traversing particle will intersect with $\mathcal{O}(1)$ sensor per layer. Sensor grouping happens based on the identification number, as well as information from the respective detector manager, called *numerology*. This identification number in Athena is specifically designed to enable this use case. The *numerology* contains information such as the number of disk layers in the endcap of a subdetector, or how many rotated sensors such a disk contains, and is populated when the `GeoModel` hierarchy is created.

In the ATLAS tracking library, the readout geometry directly builds instances of the surface base class, `Trk::Surface`, that is the basis of the tracking geometry. Conversely, the ACTS version of the tracking geometry builder has to construct the corresponding `Acts::Surface` classes itself. As the ACTS version is closely modelled after the `Trk::Surface` version, the parameter extraction and conversion works similarly.

Building of the ACTS tracking geometry begins when the configured instance of the ACTS tracking geometry service[1] initializes. The service can be configured separately whether to build the Pixel, SCT, TRT and the calorimeter (see section 7.4). Depending on the configuration, it retrieves the detector manager instances corresponding to the subdetectors. The majority of work occurs in two implementations of the `Acts::ILayerBuilder` interface. This interface describes a factory that creates layers around which the rest of the geometry is built. One implementation is used by the service for the silicon-based detectors, the other one is used for the TRT. The layer builder instances take the corresponding detector managers as their inputs.

[1] `ActsTrackingGeometrySvc`

```
namespace Acts {

class DetectorElementBase {
public:
  DetectorElementBase() = default; // trivial constructor

  virtual ~DetectorElementBase() = default; // trivial destructor

  // transform getter to be implemented by experiment
  virtual const Transform3D& transform(const GeometryContext& gctx) const = 0;

  // getter for associated surface to be implemented by experiment
  virtual const Surface& surface() const = 0;

  // getter for thickness of associated surface to be implemented by experiment
  virtual double thickness() const = 0;
};

}
```

Listing 7.1.: Interface that all detector element classes in ACTS need to implement. Most notably, it contains a function to retrieve the element transform for a given alignment context, as well as a reference to the associated surface.

Each layer builder retrieves the readout geometry elements from the detector manager and iterates over them, while wrapping each in an instance of `ActsDetectorElement`. This class implements the `Acts::DetectorElementBase` interface, which is the primary connection point between the experiment-aware geometry, and the experiment-independent view from the ACTS side. As seen in listing 7.1, the interface defines a number of virtual methods that need to be implemented by each experiment-aware detector element class. The methods allow retrieval of the element transformation matrix, a reference to a surface representation of the element, among other tasks.

The ATLAS-aware implementation of this base class accepts a readout geometry element, unpacks its shape information and constructs a corresponding surface for them. The ACTS detector element in Athena currently supports translation of the shapes shown in figure 7.3. These are rectangular, trapezoidal and line shapes. Rectangular shapes (figure 7.3a) are defined by two half-lengths h_x and h_y in the x and y directions, trapezoidal shapes (figure 7.3b) are defined by two half-lengths h_x^{max} and h_x^{min} in the x direction, as well as one half-length in the y direction h_y. The line surface (figure 7.3c) is defined by a radius r and a half-length h_z in the z direction. This choice of shapes corresponds to the shapes of barrel and endcap modules in the Pixel and SCT detectors, and the straw surfaces of the TRT.

Having obtained a list of translated detector elements, the next step is to group them into layers. This is done by using information from the coded identification number. In the Pixel and the SCT, the number of the layer that a given element belongs to is encoded. The number identifies the cylinder layers in the barrel, and the disks in the endcaps. Since the number is only unique per endcap side, the elements are grouped using a tuple of the layer number and a number identifying the endcap side.

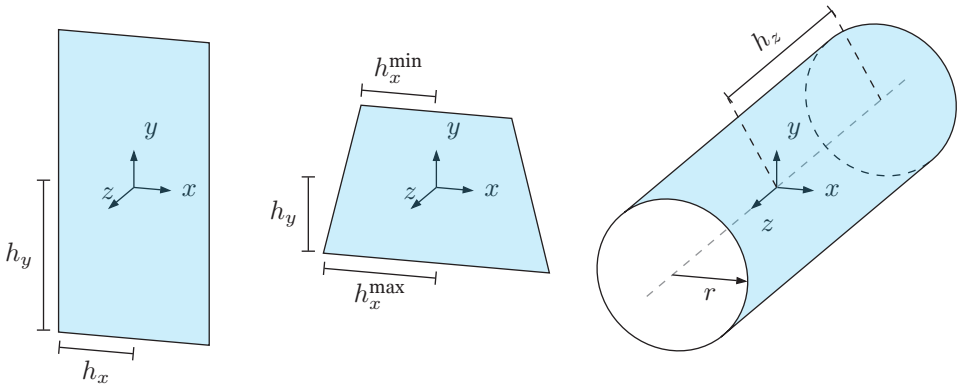


Figure 7.3.: Illustration of the surface representation for rectangular (a) and trapezoidal (b) shapes, as well as line surfaces (c). It shows the various defining parameters of the shapes, and the orientation of the surfaces with respect to a cartesian coordinate frame.

Subsequently, the layer builder iterates over the layer groups, determines a suitable extent for the layer, and assembles the required surface representations. On each layer, a binned structure is used to perform fast queries for compatible surfaces given a position on the surface representation of the layer. The binning is automatically determined by the layer builder from the locations and sizes of the contained modules. First, the routine needs to figure out what the distinct module positions are, by removing modules which are considered *equivalent* to an already existing module. By default, the routine does this by comparing the numerical module positions, which is potentially prone to errors in case of overlapping modules. Multiple surfaces can be filled into each bin, and the navigation routine also collects surfaces from neighboring bins. Therefore, some degree of variation in the binning can be recovered from, given that the binning structure roughly matches the geometric sensor structure. Automatic determination of the binning is implemented in the `SurfaceArrayCreator` class. For the ATLAS integration, the class was modified such that an arbitrary *matcher* can be supplied. This matcher must provide a signature like:

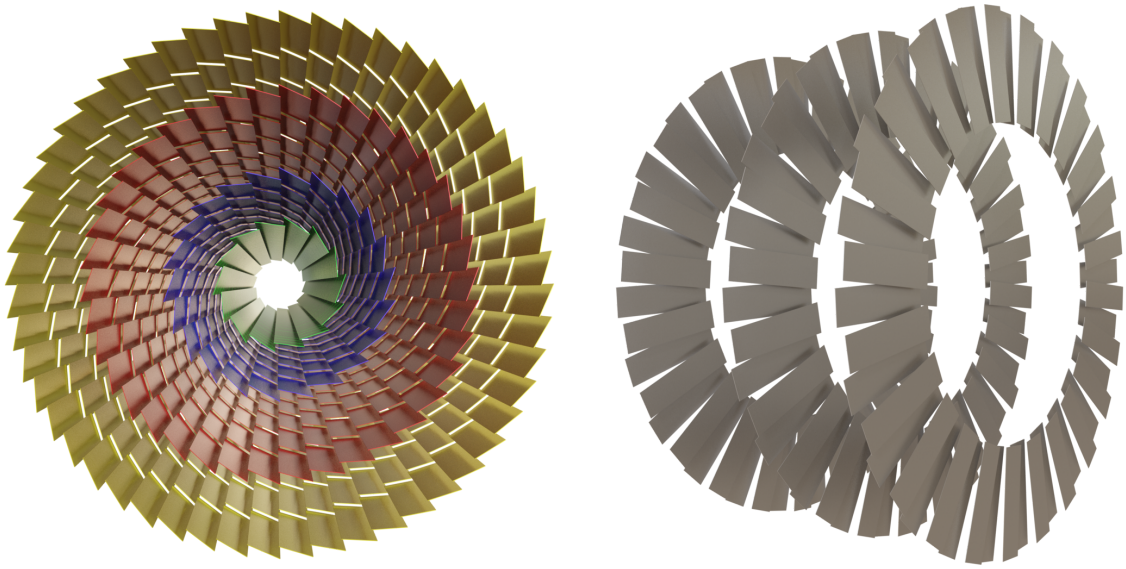
```
using SurfaceMatcher = std::function<bool(const GeometryContext&, BinningValue,
                                          const Surface*, const Surface*)>;
```

It receives the binning value that determines along which direction the surfaces should be compared. Surfaces are given as two pointers. The matcher should then return a boolean indicating whether the surfaces can be considered equivalent. Using the matcher the binning determination can be augmented with domain specific knowledge. By supplying such a matcher, the identification components can be used to make the decision if modules are equivalent. This is optimal, since the identification information is unambiguous in this regard.

The layer builders are embedded in a chain of ACTS helper classes that work together

to pack groups of layers into volumes, and then connect them in a way that is suitable for navigation during particle propagation. One such `CylinderVolumeBuilder` is initialized for each subdetector that is configured to be built. A list, ordered from the inside outwards, is then given to the top-most helper in this chain, the `TrackingGeometryBuilder`. This helper is a factory, that instructs the configured builders to produce their volumes, and connects them, effectively connecting the subdetectors. The end result of an invocation of this final helper is then an instance of `Acts::TrackinGeometry`, which contains the entire hierarchy, down to the individual sensitive modules.

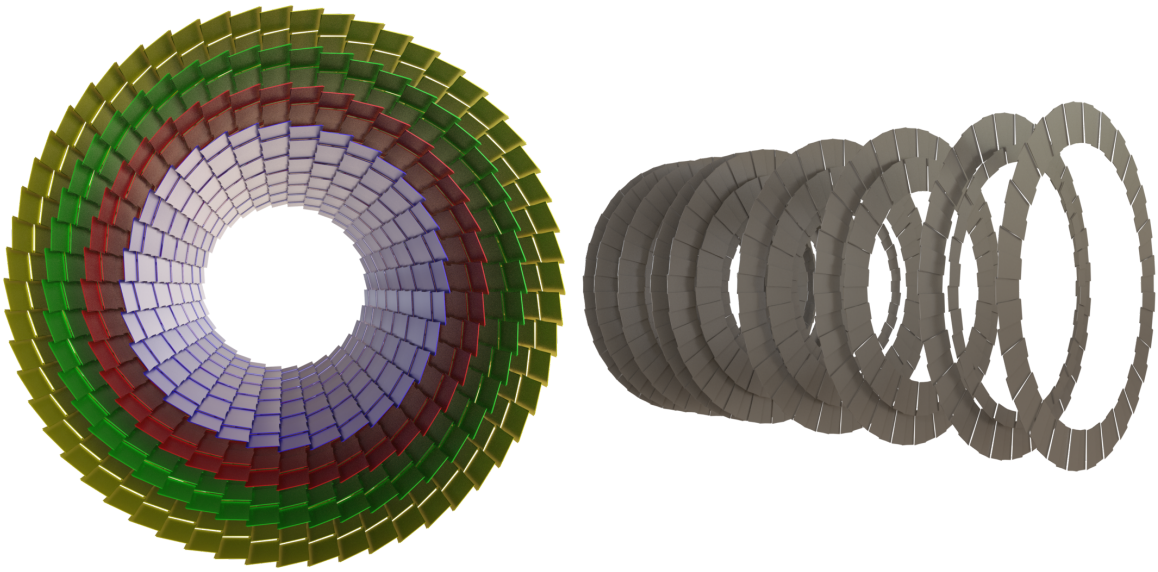
To be able to visualize the geometries, which is helpful for determining if the construction works correctly, a helper was added that is able to write out the geometry in a format that can be read by common 3D modelling software. Code with a similar purpose was previously only part of the examples provided alongside the ACTS core software. This example code is not built as part of the Athena integration, since its usefulness would be limited. The visualization parts were therefore ported directly into the `ActsGeometry` package. The functionality is exposed via a dedicated algorithm that can be scheduled in a run configuration. It will retrieve the geometry from the tracking geometry service and will write out a 3D model in the Wavefront OBJ format [209]. Subsequently, a generalized version of this visualization methodology has been integrated into the ACTS core library, to be able to more flexibly write this type of file. The visualization component deployed in Athena has yet to be migrated. In the following, three-dimensional renderings of the OBJ files are used to discuss the resulting tracking geometries.



(a) (b)

Figure 7.4.: Three-dimensional rendering of the Pixel barrel (a) and endcap (b) layers. The barrel layers are concentric cylinders, while endcap layers are disks. In (a), the different colors indicate which layer a sensor belongs to.

In the case of the Pixel and SCT detectors, displayed in figures 7.4 and 7.5, the tracking



(a) (b)

Figure 7.5.: Three-dimensional rendering of the SCT barrel (a) and endcap (b) layers. The barrel layers are concentric cylinders, while endcap layers are disks. In (a), the different colors indicate which layer a sensor belongs to.

geometry consists of concentric cylinder layers in the barrels (figures 7.4a and 7.5a) and disk layers in the endcaps (figures 7.4b and 7.5b). The barrel modules are tilted and slightly overlap in ϕ, to avoid gaps in coverage. Likewise, the endcap disks are staggered in z, where the modules again overlap in ϕ and r. The SCT modules are strip sensors, and are placed in pairs. Each pair has a small stereo angle with respect to its partner to allow a two dimensional measurement (see figure 4.13). This geometry structure directly translates to the layer model employed in the ACTS geometry description.

The TRT consists of more than 300 000 straw tubes which are not arranged in the same way as the silicon detectors. Renderings of both the barrel and endcap structures can be found in figure 7.6. In the barrel, the straw axes are oriented along the z axis. To describe this structure in terms that work in the context of the tracking geometry, multiple approaches can be chosen. The ATLAS tracking geometry logically groups straws into plane surfaces, which follow the geometric assembly. It then explicitly models these surfaces, and allows retrieval of individual straw surfaces where necessary. For the ACTS version, the decision was made to establish a baseline that does not introduce special treatment and grouping of the straws. While this model is in all likelihood not optimal, it does not require new concepts in the tracking geometry. As a consequence, the ACTS geometry models every single straw individually as a line surface during construction, and tries to group them into layers similarly to the silicon detectors. In the TRT endcaps, this model maps reasonably well: the straws are positioned in disks, so each of these disks is translated into one disk layer. However, in the barrel, there is no direct correspondence to any form of cylinder surface. To remedy this, one layer is built for each ring of the TRT barrel assembly. This sort of violates the common

(a) (b)

Figure 7.6.: Three-dimensional rendering of the TRT barrel (a) and endcap (b) straws. In the barrel, the characteristic structure of the straws can be observed. The endcaps are layers of straws radially pointing outwards from the z-axis.

assumption that layers can be modelled by *thin* surfaces, but can be counteracted by supplying a thickness value that describes the geometrical volume surrounding the assigned straws. The bin structure for both barrel and endcap layers is chosen arbitrarily to be large in the ϕ direction. To improve performance, the binning could be optimized to be as fine grained as possible without losing valid navigation targets.

Finally, a rendering of all three subdetectors of the ATLAS ID is shown in figure 7.7. It shows the Pixel detector in the center in red, the SCT around it in cyan and the TRT in green. The barrels have been cut out to reveal the inner detectors. In the case of the TRT, the endcaps have been cut out for visibility as well. It can be observed that the detectors wrap each other. In the physical assembly, they are connected by structural elements, but in this representation, there are small gaps between the various detectors. Logically, each subdetector is composed of a cylinder volume, which is again subdivided into three cylinder volumes: one for the barrel, and two for the endcaps. The cylinder volumes are synchronized such that their inner and outer radii match the corresponding opposite radii of the wrapping volume.

7.2.4. Extrapolation test

Aside from visually inspecting the geometry in a three-dimensional output that is based on individual sensors, it is also useful to probe the construction by exercising the navigation. This can be done by means of the particle propagation. A large number of particle trajectory extrapolations are launched with different directions, which are selected randomly. For each extrapolation step, information about the current state of the extrapolation is written out. Such a step is created every time a Runge-Kutta-Nyström snapshot of the particle properties

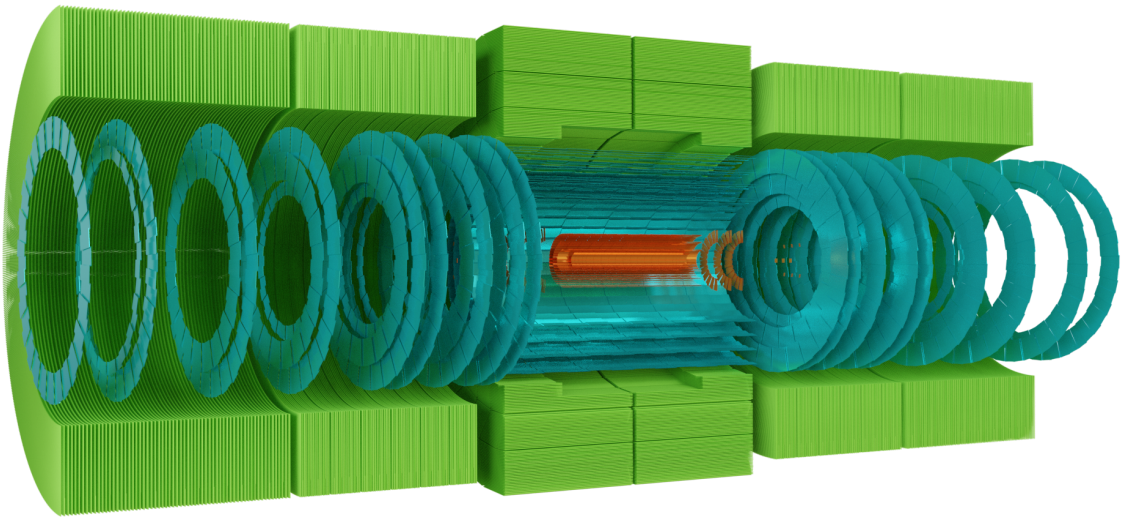


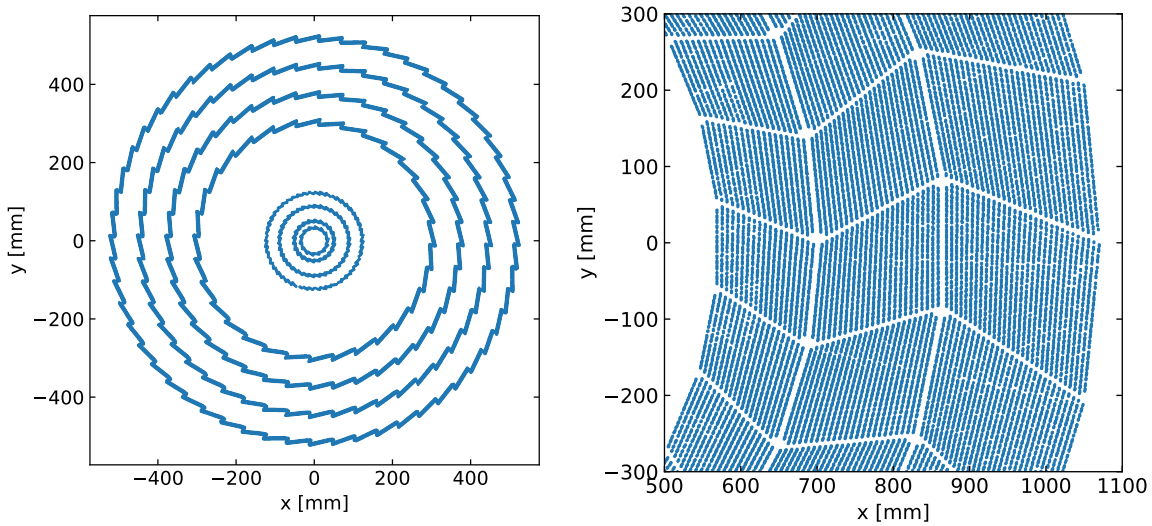
Figure 7.7.: Three-dimensional rendering of the sensitive elements in the ATLAS Inner Detector. Shown are the Pixel detector in red, the SCT in cyan, as well as the TRT in green. The detectors wrap around each other in this order, with the Pixel being the closest to the beam axis in the center of the longitudinal plane. The TRT and the Pixel and SCT barrels are cut in half to enhance visibility.

is produced. This can be in an empty volume, or be forced onto different types of surfaces by the navigation. Boundary surfaces are set as targets to navigate between volumes, layer surfaces when a layer is approached, and sensitive surfaces when an intersection with a sensor is expected. By plotting the distribution of the location of these steps, the navigation can be verified, and potential problems can be identified.

Figure 7.8 shows such a scatter plot of extrapolation steps for the ATLAS ID in the xy-plane. In figure 7.8a, for the Pixel and SCT detectors four barrel cylinder layers are shown each. The tilted sensor structure previously shown in figure 7.2 are clearly visible. Figure 7.8b shows a transverse cross-section of the TRT subdetector. Its characteristic structure, in which the straws are grouped, is resolved nicely by the propagation steps.

Aside from this, figure 7.9 shows the same ensemble of extrapolations in the rz-plane. All step locations are projected onto this plane individually, which makes it possible to see the intricate structure of the barrel and endcap layers. Also drawn here are intersections with boundary surfaces that surround each tracking volume. These intersections are drawn in orange, and are helpful to verify the navigational structure is set up correctly.

By looking at both figures 7.8 and 7.9, it is possible to visually check if the navigation and propagation works correctly. Any strong inhomogeneities here, as well as holes or missing parts of the detector indicate navigation failures. These can then usually be traced back to problems in the geometry construction, and then be fixed.



(a) (b)

Figure 7.8.: Plot showing steps of the ACTS propagation through the tracking geometry of the ID in the xy-plane. Every point corresponds to an intersection of the trajectory with a sensitive detector element. (a) depicts the silicon detectors, while (b) shows the TRT, where its characteristics structure can be observed.

7.3. Construction of ACTS tracking geometry for ITk

The previous section 7.2 described how an ACTS representation of the geometry of the current ATLAS ID is constructed. In light of the increase in luminosity of the LHC for the HL-LHC project [169], the detector systems in ATLAS and the other LHC experiments are undergoing a comprehensive upgrade. In ATLAS, the ITk [172, 173] will replace the entirety of the ID (see section 5.3.1). It will consist completely of silicon based tracking detectors, in the form of Pixel and strip detectors. At the same time, the TRT is removed to allow a larger size of the silicon-based trackers. To show ACTS' ability to adapt to this new upgraded detector, a demonstrator description of the geometry was implemented.

The implementation of the ITk detector is separated from the main branch of the ATLAS software framework. It lives in a separate branch called `21.9` that is dedicated to this type of development. The code contained there was branched off before the switch to Git and CMake with the Athena version `21`, and was originally located in a branch `20.20`. Since development on the core of the framework continued, this dedicated branch fell behind quickly, since it only received development efforts relevant to the ITk and other upgrade projects. This state was particularly insidious, since breaking changes were added to support the geometry and readout peculiarities of the ITk. Through a herculean effort, the code was ported from the `20.20` branch and CMT build system into the contemporary environment. A few incompatibilities between this version and the current version in the `master` branch still have to be reconciled in order to move forward, but the migration marks a major step in the right direction.

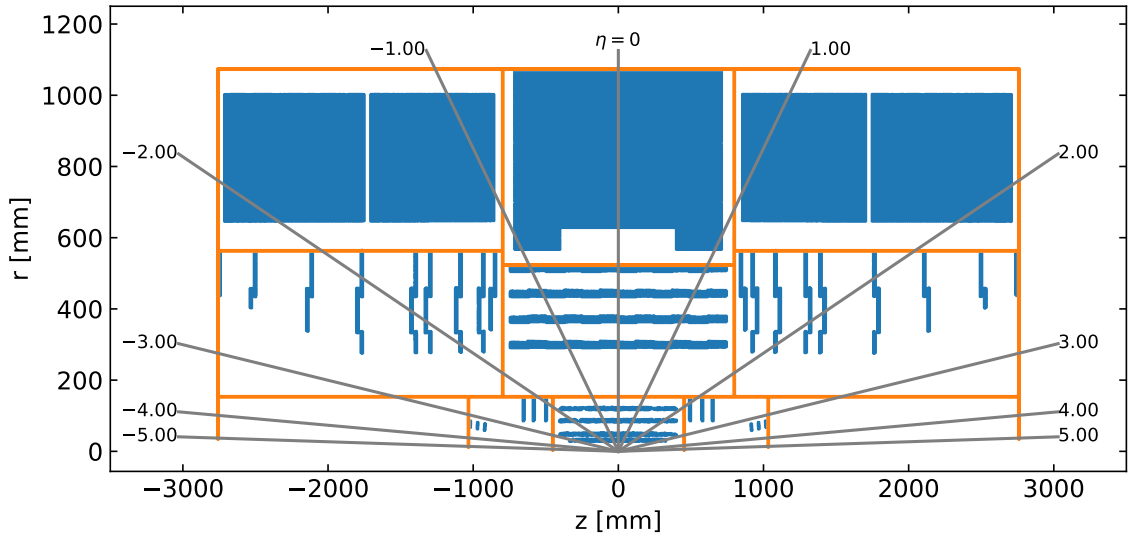
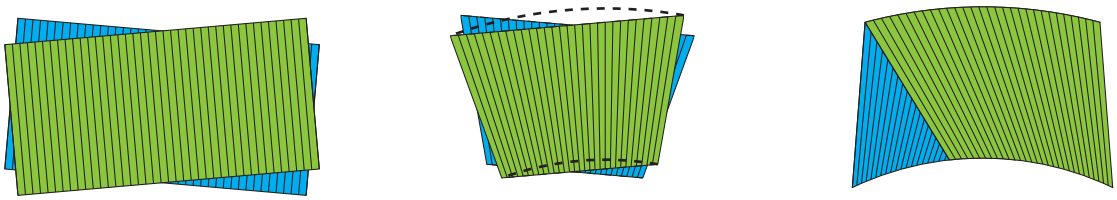


Figure 7.9.: Plot showing steps of the ACTS propagation through the tracking geometry of the ID in the rz-plane. Every blue point corresponds to an intersection of the trajectory with a sensitive detector element. Orange points can be identified with intersections with boundary surfaces.

As a proof-of-concept, a tracking geometry implementation using ACTS was implemented into the `20.20` version of the ITk software in the context of work for this thesis. The deployment strategy was significantly different from the one outlined in the previous section, where a proper component structure is defined. Since the migration work toward the `21.9` branch was ongoing, no major effort was undertaken to replicating a solid structure like this in the `20.20` code. Instead of a tracking geometry service, that builds the geometry once and provides it to clients, the tracking geometry construction helpers are directly called from a test algorithm. The algorithm is then also directly used to perform test extrapolations, like the ones discussed previously for the ID.

Since the ITk only contains silicon detectors, no code to construct a straw tube geometry was added. The same approach using the identification numbers to group sensors into their respective layers is chosen. However, instead of also using the identification numbers to determine equivalence of sensors with respect to the layer binning structure, a form of clustering was done using the sensor positions in global coordinates.

One specific addition that had to be added is an implementation of the special sensor shape that is used for the strip endcap modules. These modules are placed in pairs, as they are in the SCT, but feature a built-in stereo angle, rather than being rotated externally. This is shown in figure 7.10c, where figures 7.10a and 7.10b represent sensors with external stereo angle used in the SCT barrel and endcaps, respectively. Figure 7.10c shows the new design used in the ITk strip detector endcaps, with a built-in stereo angle. Since their shape fundamentally differs, these modules cannot be described using the trapezoid bounds that are used for the SCT endcaps, and require a more careful handling. For the geometry implementation based on



(a) (b) (c)

Figure 7.10.: Schematic drawings of different designs of strip modules with exaggerated stereo angles and dimensions. For barrel modules (a) and endcap modules without built-in stereo angle (b), pairs of modules are rotated with respect to each other to achieve a two-dimensional measurement. Sensors with a built-in stereo angle (c) do not need additional rotation.

the `20.20` branch, a rudimentary first version of such a sensor shape was implemented. This was subsequently superseded by a more fleshed out version, that is the focus of section 7.3.3.

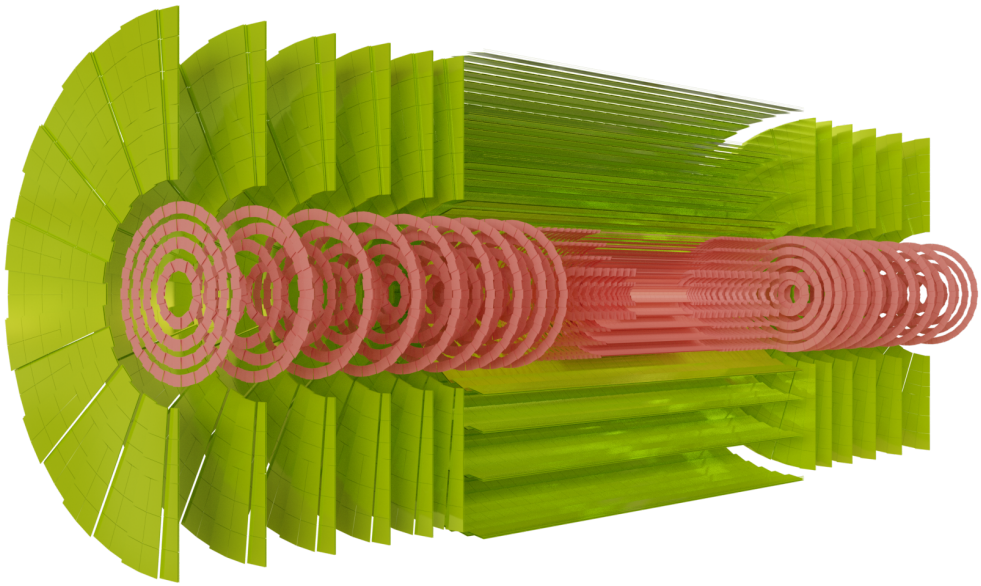


Figure 7.11.: Rendering of the ATLAS ITk detector. It shows the upgraded pixel layers in red, and the upgraded strip layers in green. A straw tube based detector is not part of the tracker anymore. The layout is the one that was present in the `20.20` in winter 2017.

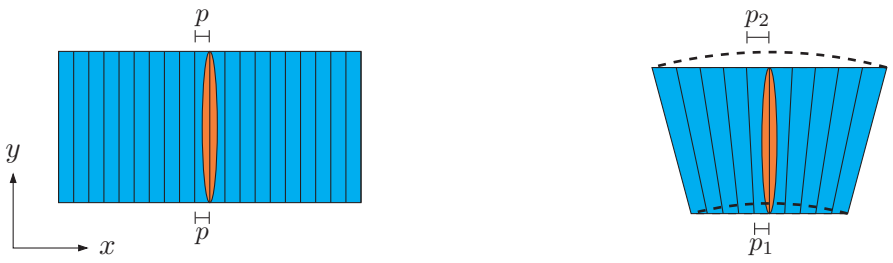
Figure 7.11 shows a three-dimensional visualization of the ITk geometry in the `20.20` branch. Note that the layout of the upgraded detector that was still being finalized when this implementation was developed. As a consequence, the layout shown in figure 7.11 differs from the one shown before in figure 5.3. The three-dimensional rendering was created by writing out an OBJ file with code very similar to the one that was eventually added to the main branch of the Athena software.

7.3.1. Review of description of endcap SCT modules

One particular issue that comes up when trying to correctly describe the ITk geometry is the handling of the new strip endcap modules. For a discussion of these, it is useful to recall the characteristics of silicon strip sensors.

Typically, these sensors are segmented in one direction, called the *pitch direction*. A variation of this is a strip sensor that is segmented in two dimensions, but the pitch in the secondary direction is much larger than in the primary one. In both cases, the spatial resolution is very good in one dimension, and there is either none at all in the other one, or it is very coarse. This means that measurements from these strip sensors are essentially one-dimensional: There is an ambiguity as to where a hit is located along the strip direction, while there is little along the pitch direction. To resolve this ambiguity, silicon strip sensors are usually mounted in pairs, in a way that their pitch directions slightly differ between each other. Exploiting this slight rotation, called *stereo angle*, it is possible to constrain the location along the strip axis and form fully two-dimensional measurement locations.

In the current ATLAS SCT, there are two types of silicon strip modules: one type in the barrel and another one in the endcaps. Illustrations of these can be found in figures 7.10a and 7.10b, which also show the strip direction. Details for both cases can be found in [94]. The barrel sensors are rectangular, and mounted such that their surface is oriented toward the beam axis, albeit with a slight tilt. This configuration and tilt can be seen in figure 7.5a on page 121. The stereo angle between module pairs is achieved by mounting them with a slight rotation around the axis defined by a perpendicular ray from the beam axis through the module center. The endcap sensors are shaped differently: they are trapezoidal. The strips on these sensors point to the center of the wheels the sensors are mounted on, which in turn is centered on the beam axis. As a consequence, the strips are not parallel. As figure 7.10b shows, they can also be rotated with respect to each other to achieve the required stereo angle. The stereo angle between pairs of modules in the SCT barrel and endcaps amounts to 40 mrad [94].



(a) (b)

Figure 7.12.: Schematic illustration of the strip layout of a barrel module (a) and an endcap module (b), and examples of cartesian error ellipses in orange. Note that for the barrel case, the strip pitch p is constant in the cartesian frame, whereas for the endcap case the strip pitch increases in the cartesian frame with higher radii (i.e. $p_2 > p_1$)

As discussed in section 4.5, during reconstruction, signals from the silicon segments are clustered. The resulting clusters are characterized by their location on the sensor, as well as their uncertainty. The uncertainty is related to the segmentation of the silicon surfaces, and is the spatial resolution of the sensor. In case of a rectangular sensor as in the SCT barrel, the covariance of a single strip is proportional to the strip length and the strip pitch, which is the distance between two neighboring strips. The covariance can be expressed as

$$C_{xy} = \begin{bmatrix} p_x^2/12 & 0 \\ 0 & p_y^2/12 \end{bmatrix}, \tag{7.2}$$

with the pitches of the sensors $p_{x,y}$ in x, y direction, and $^{w^2}/_{12}$ being the variance of a uniform distribution of width w. This covariance matrix can be visually represented as an error ellipse such as in figure 7.12. Note that, when multiple strips are combined to form a cluster, the covariance takes a modified form, as the information from multiple measurements is merged.

In the cartesian view, this direct relationship breaks down in case of non-rectangular sensor shapes. As can be seen in figure 7.12b: the strip pitch at the lower edge of the sensor is smaller than the one at the upper edge. Conversely, in a polar coordinate system, where the left and right sides point toward the origin, the pitch is constant in ϕ. There are two ways to deal with this circumstance, which are illustrated in figure 7.13:

1. Use a polar coordinate system as the local measurement frame with its center at the location where the strips would intersect. The covariance is correct in the pitch direction ϕ trivially, since this matches the physical layout of the strips. This is seen in figure 7.13a, and uses the intrinsic measurement coordinate frame, whose origin is at the center of the concentric circles defining the radial bounds.

2. Use a cartesian coordinate system and try to work around the fact that the covariance would have to grow toward larger radii by using an approximation. This approach can be seen in figure 7.13b, where the origin of the cartesian coordinate system is chosen to be the center of the module.

The difference between these two approaches was briefly discussed in section 6.3.2, and was one of key aspects being considered when designing the ACTS EDM and geometry description.

The ATLAS code uses the second approximated approach: A cartesian coordinate system is used. The origin is placed at the center of the trapezoidal shape of the sensor. The y axis points along the ”radial” axis of the sensor. To account for the pitch increase toward larger values of y, the measurement covariance is scaled to the pitch corresponding to a desired y position. Additionally, the covariance is rotated depending on the x location, such that the major axes align with the strip direction and local pitch direction, respectively.

A covariance matrix C can be transformed from one reference frame into another one by using

$$C' = J \cdot C \cdot J^T. \tag{7.3}$$

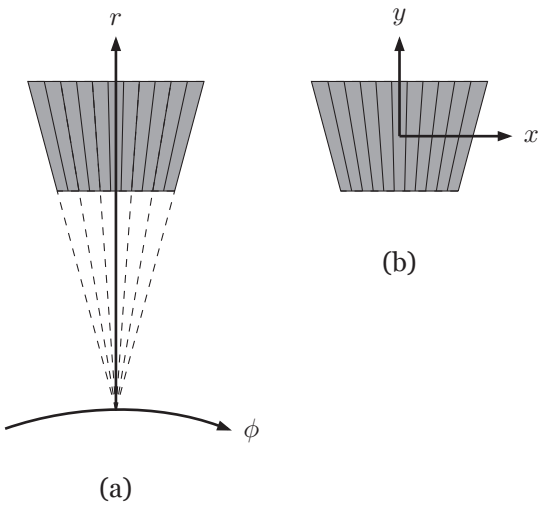


Figure 7.13.: Illustration of two ways to describe the strip sensor shapes found in the SCT. (a) shows a polar coordinate system that models the intrinsic measurement coordinate system. Its origin is the intersection point of all strips on the sensor. (b) is a cartesian coordinate system. The origin of the coordinate system is in the center of the module.

where J is the jacobian matrix of the transformation. The correction used on the clusters is a combination of a scaling and a subsequent rotation:

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \underbrace{\begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}}_{R} \cdot \underbrace{\begin{bmatrix} w & 0 \\ 0 & 1 \end{bmatrix}}_{S} \cdot \begin{pmatrix} x \\ y \end{pmatrix}. \tag{7.4}$$

In this case, the jacobian matrix is simply

$$J = R \cdot S \tag{7.5}$$

This scaling and rotation approximation is illustrated in figure 7.14, where an ellipse corresponding to a diagonal covariance matrix is first scaled up, and then rotated clockwise. Note how the third covariance ellipse's major axes are aligned with the strip and pitch directions.

In a first step, the transformation is applied using the cluster position itself. In case of a single row of strips, it is clear that no scaling takes place as there is only a single discrete possible y position. The rotation remains, however.

In addition to this transformation based exclusively on the information available from the cluster itself, the same transformation can be repeated using additional information. During track fitting, an estimation of the intersection of the particle trajectory and the sensor is available. The location of this intersection estimate can be used instead of the cluster position itself. This step is referred to as *calibration* in the ATLAS tracking context. Using the estimated trajectory position in the y direction, the effective strip pitch can be calculated and used for the scaling. This way, the expanding strip pitch toward larger values of y can be

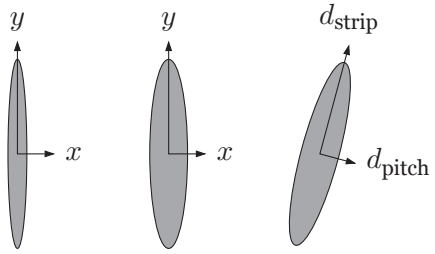


Figure 7.14.: Illustration of the scaling and rotation used to approximate the covariance associated with strip sensors in the endcaps of the SCT. On the left is the original covariance, the middle shows the covariance scaled up in the x direction, and on the right the covariance is rotated so that its major axes align with the pitch and strip direction.

incorporated into the cluster covariance even within a single row of strips. Apart from this, the procedure works identically.

7.3.2. Review of cartesian coordinate system for ITk endcap sensors

In the endcap disks of the ITk strip detector, a different type of module shape is used. Rather than externally supplying a stereo angle by rotating pairs of modules with respect to each other, a stereo angle is built into the module itself [172, 210]. This is shown in figure 7.10c and figure 7.15. These drawings show the shape in an exaggerated form to make its features more clear, but are qualitatively equivalent to the real sensor shapes. A built-in stereo angle was chosen, since the geometrical configuration of the support structure that will hold these sensors differs from the one used in the current SCT. In the ITk, it is not easily possible to rotate the modules externally and attach them to the support structure this way. So, in order to avoid having to rotate the modules, the sensor design was adjusted accordingly. In contrast to the endcap modules of the SCT, the inner and outer edges are curved. The radial bounds at r_{min} and r_{max} are consistent with an annulus shape centered around a point P_R. On the other hand, the strips point at shifted origin P_S. The two sensors in green and blue shown in figure 7.15 share the center of their *radial coordinate systems* at P_R. Conversely, the origins of their *strip coordinate systems* are different at P_S^1 and P_S^2. By virtue of having the same P_R, their inner and outer edges exactly match. It is clear that the strips of the pair of modules have a stereo angle with respect to each other, thus enabling the desired two-dimensional measurement when combining information from pairs of sensors. The stereo angle achieved using this technique in the endcaps is 40 mrad, while a stereo angle of 52 mrad [172] is achieved externally in the barrel.

The current ATLAS code in the `21.9` upgrade branch handles the shape of this module in a very similar way as the SCT annulus modules. A cartesian coordinate system is used whose origin is at the center of the concentric inner and outer circles. The x-axis points along the radial direction. To construct the cluster position and covariance, the location of the strips belonging to the cluster are retrieved. The covariance is set up as if the strip direction was

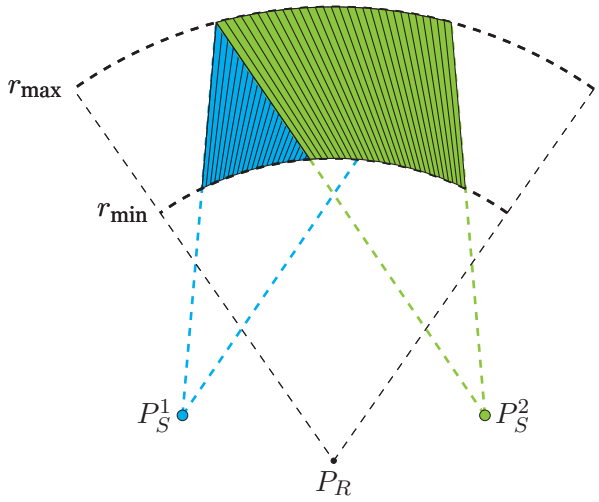


Figure 7.15.: Schematic drawing of the design of the endcap modules with built-in stereo angle. The strip coordinate system origin differs from the origin of the coordinate system in which the radial bounds are designed. The module pairs overlap perfectly in the radial direction. The distance between P_S^i and P_R, as well as the radii are not to scale, to visually enhance the characteristics of the shape.

along the x-axis. The same scaling and rotation that was discussed in section 7.3.1 is then applied to account for the strip stereo angle, and the cartesian strip pitch increasing toward larger radii.

Another aspect of the implementation is important. In the course of various track reconstruction algorithms, such as the numerical particle propagation: It is often required to test whether a point in the local coordinate system of the sensor lies within the surface bounds of the shape that describes it. This check is necessary to determine, for instance, if a particle trajectory actually intersects with a sensor, or if the intersection point on the local surface is outside of its confines. This information is interesting, for example in the context of determining the quality of tracks in an event, in order to select the ones most likely to belong to physical particles (see section 4.8 for more on this). In order not to place an overly strict requirement on tracks in light of the trajectory propagation carrying an uncertainty, there are multiple modes in which the query can be executed.

1. The simplest one is the **absolute check**, where the point strictly needs to lie inside the bounds.

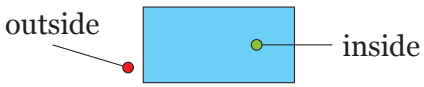


Figure 7.16.: Example of the absolute bound check for rectangular surface bounds.

2. The second mode is the **tolerance check**. In this case, a certain threshold is given along each of the dimensions of the sensor bounds, within which a point is still considered to be inside.

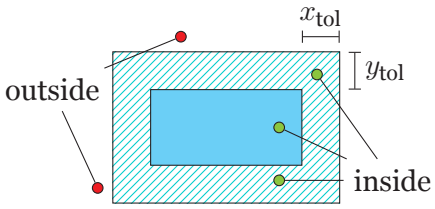


Figure 7.17.: Example of the tolerance-based bound check for rectangular surface bounds.

This check can be used to relax the absolute criterion from above by allowing a trajectory to lie slightly outside of a sensor. The tolerances ignore any kind of correlation and therefore need to be estimated from the trajectory uncertainties. A simple approach would be to use the diagonal components of an associated covariance matrix, which disregards any correlations stored in the off-diagonal components.

3. The third and final mode is the **covariance check**. This check uses the fact that often a point on the local sensor surface is associated with an uncertainty on its own. In the case of the particle propagation, this can for example be the covariance associated with the parameter prediction for the surface in question. In this case, the quantity of interest is the Mahalanobis distance [211]

$$D_M(\vec{x}) = \sqrt{(\vec{x} - \vec{\mu})^T \, C^{-1} \, (\vec{x} - \vec{\mu})}, \tag{7.6}$$

where $\vec{x}$ is test point in question, $\vec{\mu}$ is the mean position and C the associated covariance of a reference point. It can intuitively be described as the distance of two points in fractions of their standard deviation σ. The Mahalanobis distance is the generalization to multiple dimensions, and also correctly handles non-diagonal covariances. More technically speaking, it is the distance using the metric induced by the covariance. While this mode is not currently widely used, it is still very helpful to provide it.

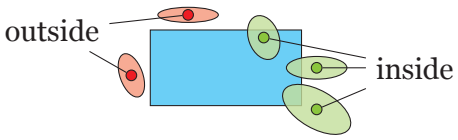


Figure 7.18.: Example of the covariance-based bound check for rectangular surface bounds. The ellipses indicate the covariances of the query points.

Since the covariance contains information on the correlations between parameters, this check can essentially directly be used with the uncertainty covariance matrix obtained from the particle propagation. Using this enables placing statistically intuitive criterions on the particle, such as it being allowed to miss a sensor by a certain number of standard deviations.

In the existing version in ATLAS, the bounds checks are implemented in the radial cartesian coordinate system. Computation of radial compatibility of a test point with the shape is trivial

in this coordinate system. To test compatibility with the sides, the edges are described as parametrized straight lines. Using the parametrization enables determining which side of the line a test point is located. Compatibility can then be tested by combining information for both edges.

For the tolerance based bounds check, the tolerances are given in the cartesian coordinates x and y. A bounding box check is conducted to enable a potential early negative return. Using the tolerances, an effective radial tolerance is calculated and checked against the radial bounds. For the edges, the tolerances are used to construct an error ellipse around the test point. If the point itself is outside the shape on a given side, the error ellipse is checked for an intersection with the edge line. If an intersection is found, the query point is considered to be inside the bounds. The usage of the error ellipse is somewhat similar to a check with a diagonal covariance matrix.

The full covariance based bounds check is implemented by describing the covariance using a, potentially rotated, ellipse. The ellipse is defined with the test point as the origin. The module shape is moved and rotated to the coordinate system defined by the major axes of the ellipse. An iterative collision test is executed against the radial bounds, and the straight edges are checked using a polygonal approximation of the ellipse as well as the module shape. Since all of these calculations are quite expensive, early returns are used to skip them when possible.

7.3.3. Polar coordinate system for ITk endcap sensors

Another approach at modelling the ITk endcap module shape was implemented as part of this thesis. This approach is based on the idea of using a polar coordinate system as the local frame rather than the cartesian one used in the ATLAS implementation. As the strips point toward one common focal point, and their pitch is constant in ϕ around it, this coordinate system aligns very well with the intrinsic measurement frame. While this describes the straight edges of the sensor nicely, the radial bounds are not easily defined in this frame. Due to this, a polar coordinate system with its origin located at the center of the circles defining the inner and outer radial bounds is also used. The origin points are the ones shown in figure 7.15 as P_R and $P_S^{1,2}$. Aside from the minimum and maximum radii and the opening angles, the class that implements this version of the shape uses the cartesian offset between these two origin points as an input. At construction, the class stores the defining parameters, as well as calculating the four corner points in both the strip and radial polar coordinate system, in addition to the strip cartesian coordinate system.

The bounds checks work differently than in the cartesian implementation. In all cases, the test points are required to be given in the strip polar coordinate system, which aligns with the measurement frame. For absolute bounds checks, the test point is trivially checked for compatibility in ϕ. Subsequently, the test point needs to be checked against the radial bounds, which is done by transforming the point into the radial polar coordinate system. In this frame, the check is again trivial.

Tolerances in ϕ and r can be supplied to the bounds check, which are assumed to be given

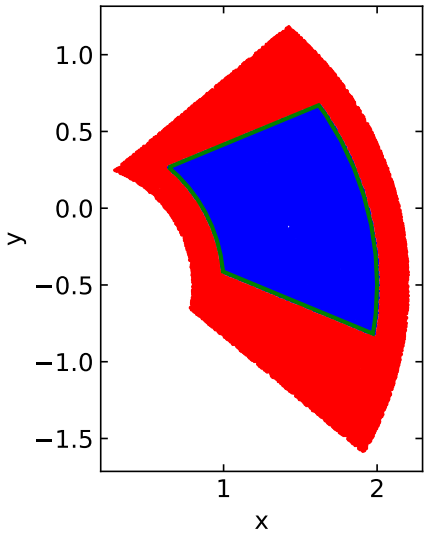


Figure 7.19.: Display of the result of the tolerance based bounds check with the polar coordinate version. Blue points are considered strictly inside the shape, red points are considered inside within tolerance. The nominal shape of the sensor is also shown.

in the strip and radial coordinate systems, respectively. Since the absolute bounds check is simply the edge case of vanishing tolerances, the same routines are used. In case of vanishing tolerances, some shortcuts and optimizations are integrated. Figure 7.19 shows a visualization of the absolute and tolerance based bounds check. It is used to visually verify that the routine works correctly. Blue points are reported as being strictly inside the shape, red points are within an arbitrarily chosen pair of tolerances around it. Test points are generated from a bivariate uniform distribution, supplied to the routine, and their results are categorized. The nominal shape of the module is also drawn. By comparing it to the overall distribution of test points to the nominal shape, it is clear that the routine works as intended.

Finally, the covariance based bounds check is implemented using the two coordinate systems as well. A test point p, alongside its covariance C_p, are inputs to this algorithm. As opposed to the tolerance based check, the covariance is assumed to be purely given in the strip polar coordinate system, as is the test point:

$$p = \begin{pmatrix} r \\ \phi \end{pmatrix}, \qquad C_p = \begin{bmatrix} \sigma_r^2 & \sigma_{r\phi} \\ \sigma_{\phi r} & \sigma_\phi^2 \end{bmatrix}, \tag{7.7}$$

where σ_r, σ_ϕ are the standard deviations along the r and ϕ axes and $\sigma_{r\phi}, \sigma_{\phi r}$ are the covariances. Given a point that is outside of the bounds in absolute terms, the covariance check can be broken down in two phases. Firstly, the closest point on the boundary of the shape needs to be found. Secondly, the Mahalanobis distance between the closest point and the test point needs to be calculated and compared to a threshold in units of standard deviations. The first step is slightly complicated by the fact that the point which the distance is calculated with,

needs to be closest in terms of the covariance induced metric. To achieve this, the test point is projected onto a segment of the boundary, using the inverse of the correct covariance as a weight. For a line segment

$$\vec{n} = \vec{b} - \vec{a}, \tag{7.8}$$

with endpoints $\vec{a}$ and $\vec{b}$, the weighted length is

$$f = \left(\vec{n}^T C_p^{-1} \vec{n}\right). \tag{7.9}$$

Correspondingly, the projection of the test point $\vec{p}$ onto the segment is

$$u = \frac{(\vec{p} - \vec{a})^T C_p^{-1} \vec{n}}{f}, \tag{7.10}$$

and finally the closest point on the line segment can be calculated as

$$\vec{x} = \min(\max(u, 0), 1) \cdot \vec{n} + \vec{a}. \tag{7.11}$$

As illustrated in figure 7.20, the calculation can be immediately done against the straight strip edges, since the covariance is already in the correct coordinate system. For the same calculation against the radial bounds, the covariance needs to be converted from the strip to the radial coordinate system.

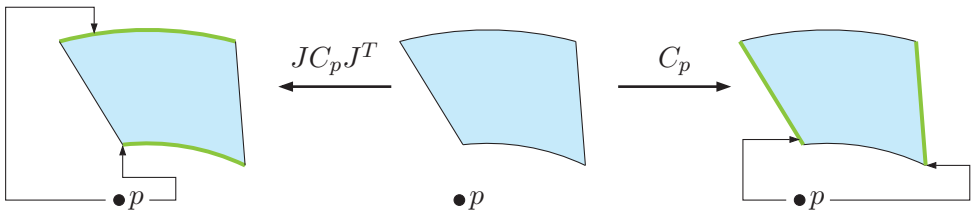


Figure 7.20.: Illustration of which coordinate system is used to project onto which bounds segments. On the left the projection onto the radial edges is shown, while on the right the projection is onto the strip edges.

Using the jacobian matrix, the covariance can be converted, as was the case for the cluster covariances, the conversion uses the jacobian matrix. Since the offset between the two origins of the coordinate systems is in cartesian coordinates, the jacobian has to include a corresponding detour. The test point position in the radial cartesian frame is expressed in terms of the origin offset $\vec{O}$ as well as r_s and ϕ_s in the strip polar frame:

$$\begin{pmatrix} x_\mathrm{r} \\ y_\mathrm{r} \end{pmatrix} = \begin{pmatrix} x_\mathrm{s} \\ y_\mathrm{s} \end{pmatrix} + \vec{O} = \begin{pmatrix} r_\mathrm{s}\cos(\phi_\mathrm{s}) + O_x \\ r_\mathrm{s}\sin(\phi_\mathrm{s}) + O_y \end{pmatrix}. \tag{7.12}$$

Subsequently, conversion into the polar radial frame can be achieved with

$$\begin{pmatrix} r_\mathrm{r} \\ \phi_\mathrm{r} \end{pmatrix} = \begin{pmatrix} \sqrt{x_\mathrm{r}^2 + y_\mathrm{r}^2} \\ 2\,\mathrm{atan}\left(\frac{y_\mathrm{r}}{x_\mathrm{r}+\sqrt{x_\mathrm{r}^2+y_\mathrm{r}^2}}\right) \end{pmatrix}. \tag{7.13}$$

The jacobian for the combined conversion from the strip polar frame to the radial polar frame reads

$$J = \frac{1}{\sqrt{A}} \begin{bmatrix} O_x \cos(\phi_s) + O_y \sin(\phi_s) + r_{strip} & r_s\left[O_y \cos(\phi_s) - O_x \sin(\phi_s)\right] \\ \frac{1}{\sqrt{A}}\left[O_x \sin(\phi_s) - O_y \cos(\phi_s)\right] & \frac{r_s}{\sqrt{A}}\left[O_x \cos(\phi_s) + O_y \sin(\phi_s) + r_s\right] \end{bmatrix}, \tag{7.14}$$

with the subexpression

$$A = O_x^2 + 2O_x r_s \cos(\phi_s) + O_y^2 + 2O_y r_s \sin(\phi_s) + r_s^2. \tag{7.15}$$

The same validation procedure using randomly generated test points can be used to visually verify the covariance routine. Shown in figure 7.21 is an equivalent picture to the one discussed before for the tolerance bounds checks. Again, blue points represent positions strictly contained in the shape, red points are below the Mahalanobis distance threshold. An arbitrarily sized covariance is used. The shape is as expected, indicating a working routine.

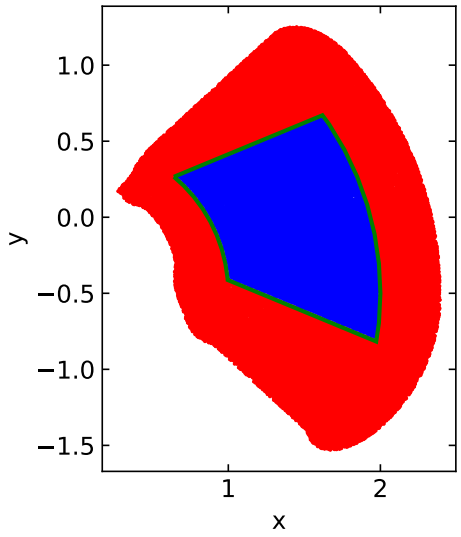
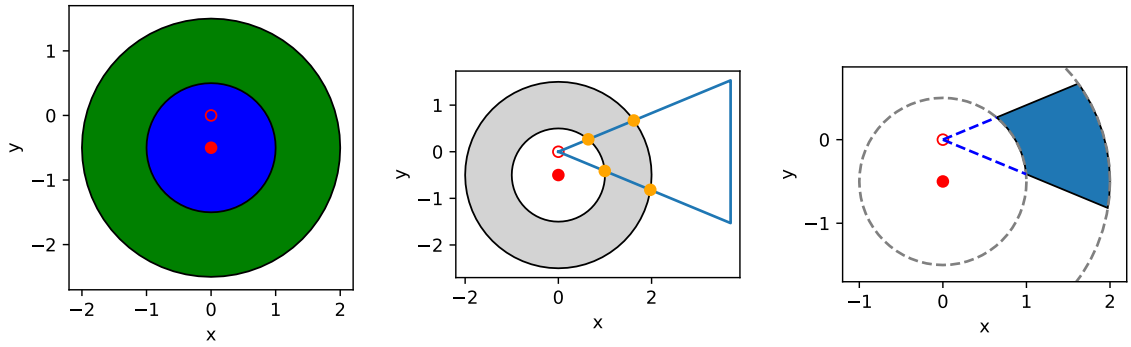


Figure 7.21.: Display of the result of the covariance based bounds check with the polar coordinate version. An arbitrarily sized covariance is used. Blue points are considered strictly inside the shape, red points are considered inside within a threshold of their Mahalanobis distance to the boundary.

All three variants of the bounds check routine were implemented in the `21.9` branch. A newly added class `AnnulusBoundsPC` complements the existing cartesian implementation `AnnulusBounds`. The class follows the general surface bound interface that is part of the

tracking library, and provides methods to perform the bounds checks. Aside from these methods, another method can calculate the minimum distance of a test point to the boundary in millimeters, rather than using the Mahalanobis distance. In addition to this, an essentially equivalent version was added to the ACTS code. This version has since been successfully used to build a standalone representation of the ITk geometry outside of the ATLAS software. Measurements of the runtime performance were taken and show that the absolute and tolerance based bounds checks complete within about 20 ns, while the covariance check takes about an order of magnitude longer. This is expected, as the computational complexity of the latter is also significantly larger.



(a) (b) (c)

Figure 7.22.: Visualization of how boolean shape compositions are used to construct the required shape. In (a) an annulus is the result of subtracting a disk (blue) from another one (green), in (b), it is intersected with a triangle shape. (c) shows the resulting shape that resembles the sensor.

The minimum distance calculation is verified by comparing its results to a reference known to be correct. Such a reference is produced using the *Shapely* [212] library in Python. As seen in figure 7.22, boolean operations are used to represent the sensor shape in the library, shown in figure 7.22c. The library provides functions that can calculate the closest point on the boundary with respect to a test point, which can be used to calculate the minimum distance. Random test points are again generated, their minimum distance is calculated using the reference. These results are then fed into a dedicated test algorithm in C++, that constructs the same shape from the provided defining parameters, and also calculates the minimum distance. No discrepancy between the reference and the implementation are found, leading to the conclusion that the calculation is working correctly.

As a last step, conversion from the cartesian to the polar coordinate implementation is enabled by a static factory method that takes an instance of `AnnulusBounds` as its input. It extracts the defining parameters, and converts them into the cartesian offset shift and opening angles ϕ in the strip system. This conversion, and more generally the consistency of the cartesian and the polar coordinate versions, is verified as well. Random defining parameters of the cartesian version are drawn and used to construct instances. Each instance is converted into a corresponding polar coordinate version. This includes converting coordinate systems

such that the cartesian shape is located on a plane surface, and the polar coordinate shape is located on a disk surface centered around the strip focal point. The location and rotation of the surfaces are also chosen at random. Following the translation, for each discrete shape, a number of test points is drawn in the local coordinate system of the plane surface. Each point is converted to global and then to local coordinates of the disk surface. Both local coordinate representations are then fed into the bounds check routine, and the result is returned. A visualization of one generated instance is shown in figure 7.23. Green points indicate agreement, red points indicate a discrepancy. No red points are observed. The polar coordinate implementation is found to yield identical results within floating point precision, no discrepancies are observed. More visualizations of a larger number of randomly generated shapes can be found in figure A.1.

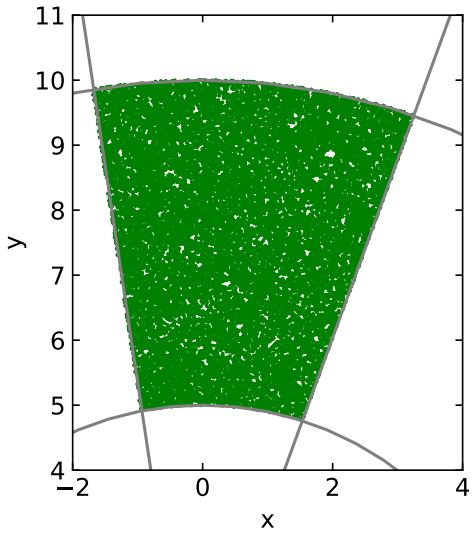


Figure 7.23.: Visual comparison between the existing cartesian bounds and the new polar coordinate bounds. Test points are drawn in green if both implementations agree whether it is inside the shape, otherwise they are drawn in red. No red points are visible.

To complete work on the implementation, the applicability of the polar coordinate based description of the ITk strip endcaps in the actual track fit was tested. The hypothesis here was that during a Kalman Filter pass, the different parametrization should lead to equivalent results if the cluster position and uncertainty are consistently described. Performing a full-fledged verification comes with the complication that the entire chain would have to be modified, where the geometry construction of the ATLAS from the data source directly builds the disk surfaces and polar coordinate bounds. Aside from this, all other clients of the geometry would have to be updated to be able to handle the parametrization. Avoiding this complete migration for the purpose of verification alone, another approach was followed. During ambiguity resolution, a stage where the reconstruction tries to remove duplicate or bad tracks by applying quality criteria and ranking them, the new implementation can be injected.

This involves three steps. First, for every instance of a strip sensor, a corresponding polar coordinate version has to be built. These instances can be cached to avoid rebuilding them frequently. They live outside of the tracking geometry, and are associated via maps. Then, during the actual ambiguity resolution, the `ForwardKalmanFitter` class is invoked, and tasked with fitting a sequence of measurements. This happens to re-estimate the track parameters after having removed measurements due to overlap with other tracks, among others.

This fitter tasks the `SCT_ClusterOnTrackTool` to perform the calibration of the measurement given the current predicted parameter estimate. Calibration was discussed previously, and the default implementation performs the scaling and rotating of the cluster covariance in this phase. Test code was added here to instead consume the already created cartesian cluster, go back to the raw data source, and build a polar coordinate cluster instead. In this case, the calibration can be dropped, since the coordinate system handles the ϕ pitch correctly already. After the ambiguity resolution completes, the collection of found tracks is written to an output file.

To study possible differences, outputs with and without the modifications mentioned above were created, and compared. Some minor differences relating to the residuals could be observed. Additionally, one of the quality criteria of the tracks, the number of outliers, was studied. Here, a measurement which is too far away from the position that is expected from the particle propagation is considered as an outlier. However, only changes that are small enough to be attributed to potential discrepancies between preceding stages of the reconstruction were found. Due to the fact that those stages and their parameters are closely tuned with the cartesian implementation as a reference, some degree of disagreement is likely to be expected. Since overall track parameters such as resolutions themselves seem unaffected, this discrepancy therefore does not raise concern. A full implementation would have to include all the changes mentioned before, and produce a comparison where tuning toward the polar coordinate measurement frame has taken place as well. This effort has yet to be undertaken.

7.3.4. Summary

This section described the work toward modelling geometry of the Phase 2 upgrade of the ATLAS experiment for the HL-LHC. After introducing general aspects of the ITk construction and layout, at first, a feasibility study into whether the overall geometry can be described using ACTS. This study, carried out in the discontinued `20.20` branch in 2017, showed that no general issues arise when building an ACTS tracking geometry version of the ITk.

One aspect needed further work, however. The endcap modules of the ITk strip system feature a new design which uses a stereo angle built in to the shape of the module, rather than rotating pairs of modules with respect to one another. Aside from that, it features rounded edges at its inner and outer radii, making the trapezoidal model that was used in the SCT suboptimal. After a discussion on commonalities and differences between this new design and the existing design of the SCT endcaps, as well as the previously existing cluster covariance calculation, a new approach was presented. Using two polar coordinate systems to describe the new design, most of the issues can be resolved. One matches the radial edges

of the shape, which are segments of concentric circles around the beam axis. The other one matches the strip coordinate system, whose origin is shifted away from the beam axis, to achieve the desired stereo angle. This coordinate system coincides with the intrinsic strip measurement reference frame, and can be used to cluster locations and uncertainties in the form of covariance matrices. Details on the implementation of boundary checks with this new design with absolute, tolerance and covariance based criteria were given.

To verify that the polar coordinate system can be used for the clusters, an integration test with the ATLAS cluster building and calibration was conducted. By converting clusters from existing tracks from the cartesian coordinate system to the new polar coordinate system, the fitting infrastructure was tested. The KF algorithm responsible for refitting selected tracks to extract the best parameter estimates was executed on the modified clusters. Fitting performance was in line with expectations, and no roadblocks were encountered.

A full scale migration of the cartesian description to the new polar description has yet to be done. This, however, requires a significant amount of changes in a vast portion of components contributing to the cluster production, conversion, handling and fitting.

7.4. Implementation of a fast navigation system through arbitrary geometries

Calorimeter systems serve the purpose of measuring the energy of particles. While they often feature segmentation to give some spatial resolution to energy deposits, reconstruction is usually done differently as is the case for tracks (see section 3.2.3). While calorimeters are evolving toward higher granularity, such as for the CALICE study or the CMS Phase-2 calorimeter upgrade, in most cases, reconstruction of calorimeter systems is a clustering problem, rather than a combinatoric one. Reconstruction algorithms therefore typically differ substantially. Nevertheless, a description of calorimeters in the context of a tracking geometry can be useful, as is outlined in the following.

Firstly, as is the case in ATLAS as well as many other experiments, the Muon Spectrometer (MS) is located outside of the calorimeter systems. This means that particles have to traverse the calorimeters before reaching hardware for muon detection. In turn, this requires any particle propagation to be able to traverse the calorimeter volumes, for example to match ID muon candidates to MS tracks. Here, information on the material distribution is required to be able to correctly make predictions on the particle properties after traversal. This can be done in a few ways, one of which is a continuous material effect integration as briefly touched on in sections 4.3.3 and 6.5, paired with volume based information on the material. This approach is currently under development in ACTS. Another approach is using conventional virtual material layers to approximate the distribution of passive material, and then using the discrete integration of material effects. Aside from this aspect arises the question of navigation. If there is no interest in the substructure of the calorimeters, they can simple be modelled as passive volumes.

There are, however, some cases where this substructure information is relevant. One such

case is the application of particle flow algorithms for jet reconstruction. In this context, information of which calorimeter cells a propagated particle would have hit can be used when combining track information with calorimeter cluster information.

7.4.1. Conventional navigation model

To understand how modelling a calorimeter geometry differs from a tracking detector structure, it is helpful to discuss the general approach to geometry navigation employed in both ATLAS and ACTS in a bit more detail. As mentioned in section 4.4, the geometry is described as a hierarchy of volumes. The highest object in the hierarchy is a global *world* volume. At the lowest level, volumes contain only a number of sensors. Volumes can also contain layers, which represent a collection of sensitive surfaces or locations of material, where the assumption is that a typical traversing particle will only intersect at one point.

The navigation will look for layers in the volume that contains the current location of the propagation. It will then check found layers for general compatibility with the particle trajectory, usually by means of an intersection with a straight line. If compatible layers are found, the navigation resolves these layers, which means that it looks for compatible sensitive surfaces contained in them. This is again done by intersection with a straight line, which is usually a sufficient approximation if a search window is used for finding compatible surfaces around the intersection point. The navigation will then instruct the propagation to attempt approaching each of the compatible sensitive surfaces. When this is done, the navigation will revert back to the layers, and continue with the next one. Once all layers are handled, the navigation is complete in the current volume.

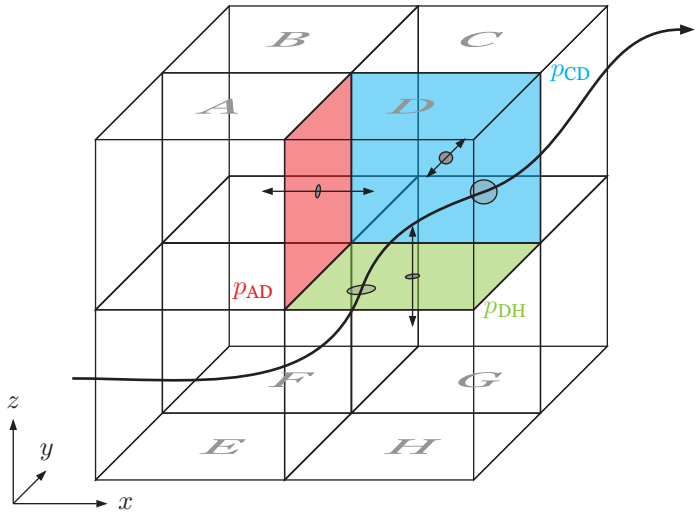


Figure 7.24.: Illustration of the conventional navigation model used in the ATLAS tracking software, as well as ACTS. Shown are eight cube volumes that are connected via there interfaces, called *boundary surfaces*. A particle trajectory is shown that intersects two of them, which illustrates the navigation model following boundary surfaces between volumes.

Subsequently, the navigation will try to find the volume that should be handled next, in

a way that is shown in figure 7.24. The navigation will attempt to calculate intersection points with the boundary surface of the current volume. For the example volumes A-H in figure 7.24, three boundary surfaces p_{AD}, p_{CD} and p_{DH} are shown. They are the interfaces between volumes A&D, C&D and D&H, respectively. The example trajectory starting from the lower left crosses from H into D through p_{CH}, before migrating to D via p_{CD}. To support the navigation, boundary surface holds pointers to the volumes it connects, which effectively *glues* the volumes together. This way, the boundary surfaces essentially act as portals between the different volumes.

Looking at the geometry of the ATLAS calorimeter system is instructive to discuss the challenges applying the conventional navigation model to complex geometries. Figure 7.25 shows a three-dimensional illustration of the *readout geometry* of the ATLAS calorimeter systems. Shown are the Liquid Argon (LAr) calorimeter in green, the tile calorimeter in red, as well as the Forward Calorimeter (FCal) in blue. These three systems use differing underlying detector technologies and feature varying segmentation. In particular the LAr calorimeter is divided into a large number of cells as a function of η and ϕ. In the physical hardware, this segmentation is designed to enable the calorimeter to localize energy deposits. The representation shown here, does not correspond exactly to the physical hardware, however. The location of the cells in this representation is designed to best model the location that energy deposits should be associated with. Physical cells can be associated to more than one cells in the readout representation, and signals can be split. Overall, the readout calorimeter geometry comprises almost 190000 individual cells.

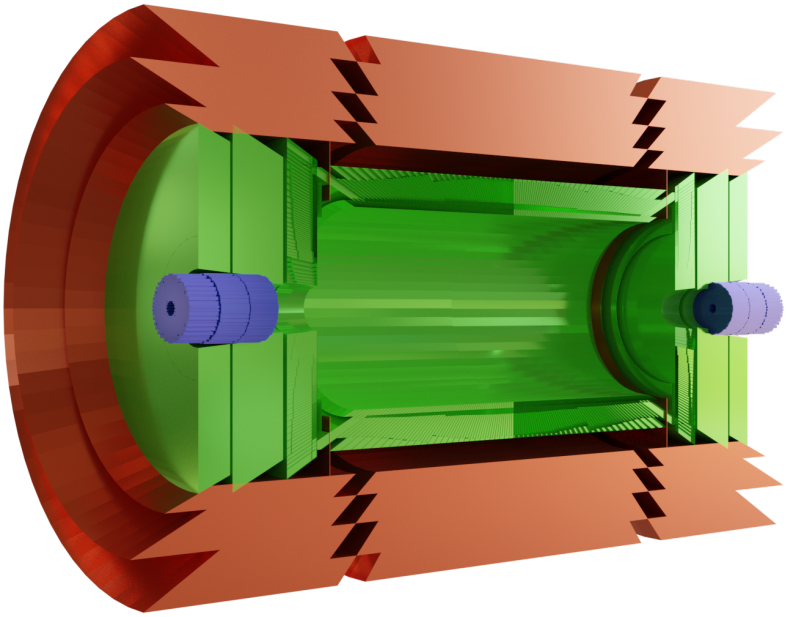


Figure 7.25.: Readout geometry representation of the ATLAS calorimeter. Shown are the LAr calorimeter in green, the tile calorimeter in red and the FCal in blue.

To accommodate this geometry with the navigation model described above, every cell is

modelled as a volume. The geometry builder then needs to walk through all cells and connect them using boundary surfaces. In case a cell's face touches more than one other cells, the boundary surface needs to be split, such that it can contain more than one *portal* definition. This approach to construction and navigation is implemented in th ATLAS tracking software to build a representation of the calorimeters. It is apparent that it relies on detailed knowledge of the layout, requires hand-tuning the construction such that boundary surfaces are set up correctly, and is potentially brittle when the geometry changes.

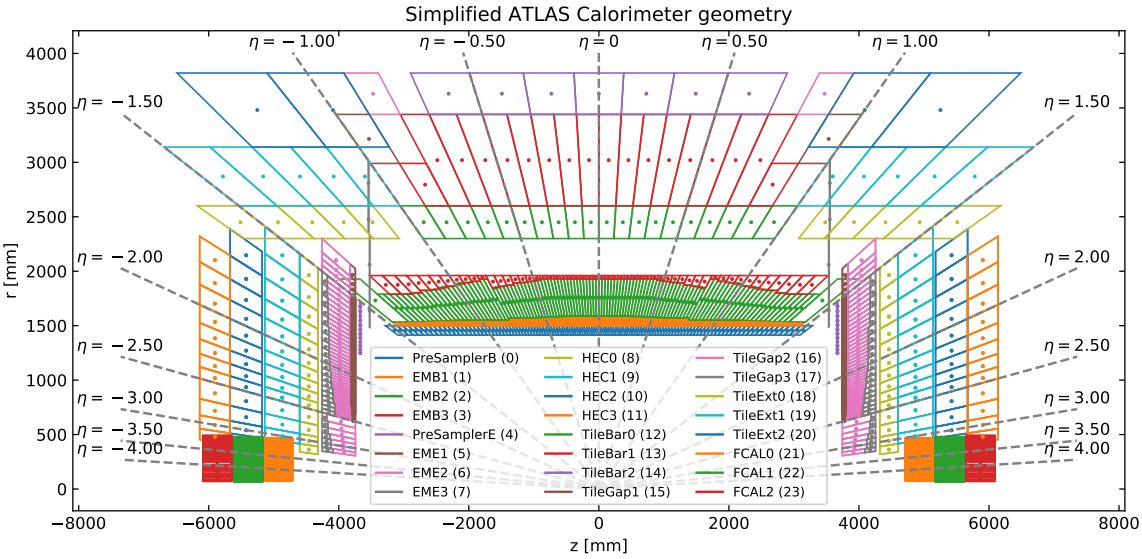


Figure 7.26.: Schematic cross-section of the rz-plane through the calorimeter readout geometry. Shown are the individual cells that are modelled in the geometry. The legend categorizes the various cells into logical groups.

When looking closely at figure 7.25, it is possible to spot overlapping cell volumes. While this obviously is not the case in the actual hardware, it improves the modelling of the logical location of signals in the readout geometry. These overlaps can be understood by considering that this geometry model is not a direct representation of the physical hardware. Rather than trying to model the individual physical components, virtual logical cells can combine multiple physical calorimeter cells. The sizes and exact locations of cells are optimized such that a readout associated with it best matches the group of underlying physical cells. In some cases, this leads to the logical volume of a group of cells overlapping with another one.

The same overlaps can also be observed in figure 7.26, which shows a cross-section of the readout geometry in the rz-plane. Individual cell center positions are shown, as are the boundary surfaces. It is clearly visible that the cells in the LAr and tile calorimeters are segmented along η. The FCal's segmentation is oriented with the global cartesian axes. The categorization of all cells is indicated by their color. In this view, some additional overlapping volumes become visible. It is clear that the conventional model described previously breaks in the presence of overlapping volumes. Since boundary surfaces need to be defined unambiguously, the navigation cannot handle the case where volumes intersect. A possible

solution would be to subdivide volumes in order to resolve overlaps, which further increases the complexity of the requires implementation.

To resolve this issue and simplify modelling and handling of geometry structures like the one present for the ATLAS calorimeter, a generic solution was developed in the context of this thesis, which is discussed in the following sections.

Ray tracing is a concept widely employed in computer graphics. It describes the process of casting rays from a source in a specified direction. Subsequently, intersections with geometric objects are calculated. To generate an image, a large number of rays are *emitted* and traced. Depending on the implementation, rays are either emitted from the viewer into the pixels of the viewport, or from light sources in the scene. At each intersection, the ray direction is modified according to the physical process of reflection. Intersections and reflections are repeated until either a maximum number of reflections is reached. In the case of emission from the viewer, tracing also terminates when a light source is reached, or vice versa. At this point, the pixel color can be evaluated using the light path and intersected objects as an input. Modern models use physically based lighting formulae, and incorporate refraction, roughness and reflectivity of material, shadows and reflection of objects in other objects.

Due to its popularity, a lot of effort has been poured into making each part of the procedure as fast as possible. Best practices and insights from such efforts can be used to implement fast intersection tests with arbitrary geometries. The idea is to describe the particle trajectory in a simplified way, and use that to resolve possible intersections with volumes in the geometry. Being a straight line, a ray can trivially be used to describe neutral particle trajectories. Additionally, they can approximate charged particle trajectories, given that their momentum is either large enough to not introduce excessive curvature due to magnetic fields, or the magnetic fields are sufficiently low. In case curvature needs to be factored into the navigation, it would seem reasonable to use a cone to build an envelope of possible curvature. Figure 7.27a shows how such a cone can form an envelope around a trajectory, covering a certain maximum amount of curvature. While a cone is conceptually natural, computationally it is less than optimal, as will become clear shortly. Instead, a certain type of *frustum* can be used to construct an envelope for a maximum curvature.

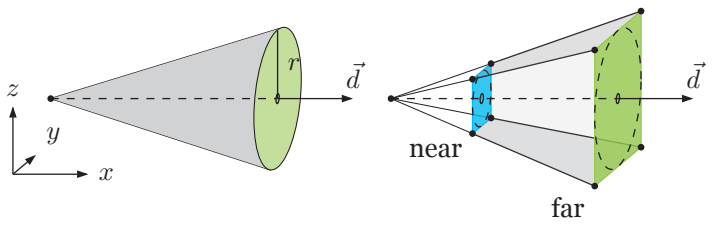


Figure 7.27.: Illustration of a cone around a straight trajectory (a), as well as a corresponding frustum that approximates the opening angle of the cone (b).

Generally, a frustum is the remainder of a solid if it is cut by two parallel planes. In the context of computer graphics, a frustum is frequently used to define the view of a scene. The

two defining planes are commonly referred to as the *near* and *far* plane and they are combined with a pyramid to define the *view frustum*. Figure 7.27b shows a four-sided frustum, and how it can approximate a cone of a given opening angle. As is obvious from the illustration, the area of the far plane is larger than the circle of a cone with the same opening angle. This excess can be reduced by increasing the number of sides for the frustum. For the purpose of approximating a cone with a frustum, any rotation around the direction vector $\vec{d}$, also shown in 7.27b can be ignored. The approximation remains effectively identical, as can be seen in figure 7.28

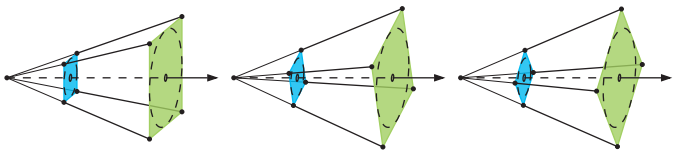


Figure 7.28.: Illustration of how the rotation around the direction vector does not affect the approximation of the underlying cone by a frustum.

7.4.2. Bounding boxes

The implementation of the navigation uses the objects from above to find intersections with the detector geometry to provide potential targets. For this, algorithms for calculating these intersections are required. Computing intersections of rays and frustums with arbitrary shapes is highly nontrivial. It could be accomplished by providing intersection algorithms for each shape, then looping over volumes and dispatching to the correct algorithm for the corresponding shape. This approach, however, does not feature the performance characteristics desirable for a fast navigation module, due to looping over a large number of objects, and having to perform dynamic dispatch very frequently. Instead, the various possible shapes are approximated by a homogeneous shape that is simpler to compute intersections against. For this, every shape is replaced by a corresponding *bounding box* that contains it completely. There are various different types of bounding boxes that fulfill this general requirement. They differ in complexity of construction and intersection calculation, as well as how well they match the wrapped shape.

The conceptually simplest bounding box is the Axis Aligned Bounding Box (AABB). Such an AABB is shown in figure 7.29a wrapping an arbitrary shape. It is defined by the maximum extent of the wrapped shape along each cartesian axis in the global frame. Due to being aligned to the axes, it can be represented by only two points, the minimum and the maximum vertex. AABBs feature the best intersection performance and have a low memory footprint, but only loosely wraps the underlying object, depending on its orientation, as seen in the figure.

A variation of the AABB is the Oriented Bounding Box (OBB), which is aligned with the local coordinate frame of the shape rather than the global one. An example for an OBB is shown in figure 7.29b. The minimum and maximum vertex displayed here are only valid if the coordinate system is rotated such that the edges of the OBB align with the axes. By choosing an optimal rotation, the OBB can provide a much closer match to the wrapped

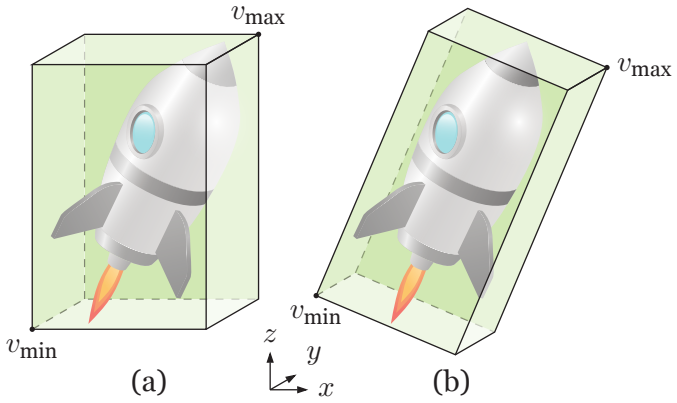


Figure 7.29.: Illustration of two bounding box types containing an example object. (a) depicts an Axis Aligned Bounding Box, whereas (b) shows an Oriented Bounding Box. Also shown are the minimum and maximum vertices, in the global frame for the AABB, and in a rotated frame for the OBB.

shape, as is apparent when looking at the figure. However, in addition to the two defining points, the rotation matrix from the global to the local frame needs to be stored. By combining this rotation matrix with the data of an AABB, an OBB can be represented with the same implementation.

An implementation for AABBs was added to ACTS which is generic over the number of dimensions. It contains a link to an arbitrary object that can be used to associate a bounding box with a contained shape. In ACTS, volumes are described by a generic class `Acts::Volume` that contains a volume bounds instance inheriting from `Acts::VolumeBounds`. The volume instance contains information on the local coordinate frame of the volume, as well as the actual shape. The shape is assumed to be located at the origin of the coordinate system, a transformation matrix is stored at the volume level. Functionality to construct a bounding box was added to all volume bounds classes. In the local coordinate frame, the AABB and OBB of the volume are identical. Each volume creates an AABB that is implicitly identical to an OBB at construction using this method and stores it for later use. Using the volume's transformation matrix, the OBB can be converted into an *actual* AABB. Here, all eight corner vertices of the OBB need to be converted and the maximum extent has to be calculated in global coordinates to derive the AABB.

7.4.3. Ray and Frustum intersections with boxes

Instead of implementing intersection algorithms with every possible shape, the navigation now only needs to be able to calculate intersections with these bounding boxes. The following discusses these algorithms for intersections with AABBs, however, they can be reused for OBBs by rotating the respective object into the OBB coordinate frame.

Ray-Box intersection

Intersections between rays and AABBs can be calculated using the so-called *slab-method* [213]. It is convenient to illustrate the algorithm in two dimensions, and it can trivially be generalized for three dimensions. The algorithm is widely used in ray tracing applications, and is optimized to include as few computationally expensive operations as possible.

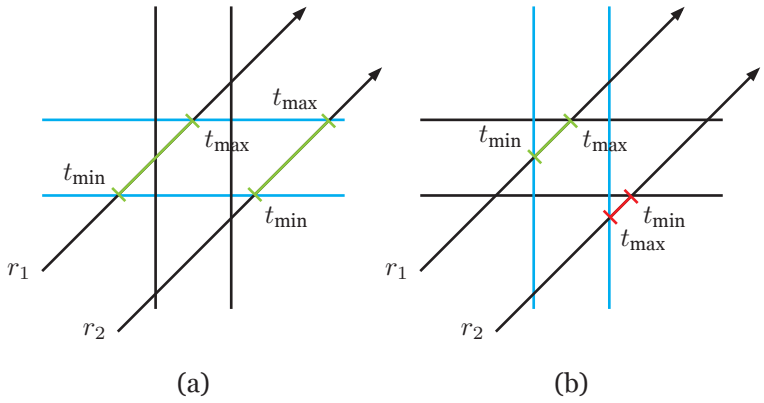


Figure 7.30.: Illustrations of the steps of the slab-method algorithm which calculates whether rays $r_{1,2}$ intersects with an AABB in two dimensions. In (a), intersections with the bounding box planes in the x dimension are calculated. In (b), the intersections with the planes in the y-direction are calculated.

Figure 7.30 shows a visualization of the algorithm for two rays r_1 and r_2. It works by calculating the distances along the ray direction at which the ray intersects with the planes defining the bounding box. Calculating these distances is performed one dimension after the other, for each dimension two intersections are calculated. In figure 7.30a, distances along the rays t_{min} and t_{max} are calculated by intersecting the planes in the y directions. At this stage for both rays $t_{\text{min}} < t_{\text{max}}$ holds. Subsequently, the intersections with the planes in the x direction are calculated, as seen in figure 7.30b. For r_2, the new value for t_{max} is smaller than the previous one, so it is stored. The minimum (maximum) value is only overwritten, if the current value is larger (smaller) than the one from previous dimensions. In the specific example in figure 7.30b, the new t_{max} is smaller than the previous one for r_2, so it is stored. If after processing all dimensions

$$t_{\text{min}} < t_{\text{max}} \tag{7.16}$$

holds, the ray intersects the bounding box, otherwise it does not. In the case depicted in figure 7.30, the condition from equation (7.16) evaluates to true for r_1, while it evaluates to false for r_2: r_1 intersects the box, while r_2 does not.

Listing 7.2 shows C code that implements the aforementioned slab-method algorithm in two dimensions. It uses two structures, `box` and `ray`. The former contains the two defining vertices of an AABB, the latter contains the starting position and direction of a ray. The code first calculates the intersection points of the ray with the x components of the defining vertices along the ray direction. The minimum and maximum of these points is then assigned to

`tmin` and `tmax`. In the first dimension this happens unconditionally. Subsequently, the code performs the same calculation against the y components of the defining vertices. It now stores the minimum and maximum if the new values are larger or smaller than the previous ones respectively. The final line compares `tmin` and `tmax` and provides the result.

```
bool intersection(box box, ray ray) {
  double tmin = -INFINITY, tmax = INFINITY;

  for(size_t i=0;i<3;i++) {
    if (ray.dir[i] == 0.0) continue;
    double t1 = (box.min[i] - ray.origin[i])/ray.dir[i];
    double t2 = (box.max[i] - ray.origin[i])/ray.dir[i];
    tmin = std::max(tmin, std::min(t1, t2));
    tmax = std::min(tmax, std::max(t1, t2));
  }

  return tmax >= tmin;
}
```

Listing 7.2.: An implementation of the slab-method for calculating an intersection between a ray and an AABB. The function takes a `box` and a `ray` instance as input, and returns whether they intersect. [214]

The code example from listing 7.2 can be optimized further. The ray direction components are divided by twice, so an obvious improvement is to calculate the inverted values once and replace the division with a multiplication. This is especially helpful if the ray is not only intersected with a single AABB, but with multiple ones. In this case, it is convenient to store the inverse direction alongside the direction in the ray structure. Care has to be taken that the direction components can always be converted, which is not trivially the case for vanishing components. To extend the algorithm to three dimensions, a third block needs to be added. Using some details of the way floating point operations are carried out by CPUs, the `if` blocks and corresponding branching instructions can be eliminated, which improves performance. Some care has to be taken to make sure that the edge case where ray lies *exactly* on one of the planes of the AABB is handled correctly [215]. Table 7.2 shows micro benchmarks of the effect of the optimizations carried out. Calculating the inverse once instead of twice reduces runtime per intersection by about 40 %, while further optimizations remove about another 20 % of runtime relative to the unoptimized code from listing 7.2.

Finally, the algorithm can be made generic over the number of dimensions without having to resort to an explicit loop over a runtime value of the number of dimensions by using operators provided by the `Eigen` library. The code that was added to ACTS in order to implement this algorithm can be found in listing 7.3. It is generic over the number of dimensions, and `Eigen` is made aware of this number at compile time, allowing further automated optimizations. The correctness of the intersection algorithm is verified by unit tests that test a number of rays for intersections with a pre-defined set of bounding box, and compares the result to the known correct one.

```
template <typename entity_t, typename value_t, size_t DIM>
bool Acts::AxisAlignedBoundingBox<entity_t, value_t, DIM>::intersect(
    const Ray<value_type, DIM>& ray) const {
  const VertexType& origin = ray.origin();
  const vertex_array_type& idir = ray.idir();

  // i
  vertex_array_type t0s = (m_vmin - origin).array() * idir;
  vertex_array_type t1s = (m_vmax - origin).array() * idir;

  // ii
  vertex_array_type tsmaller = t0s.min(t1s);
  vertex_array_type tbigger = t0s.max(t1s);

  // iii
  value_type tmin = tsmaller.maxCoeff();
  value_type tmax = tbigger.minCoeff();

  /// iv
  return tmin < tmax && tmax > 0.0;
}
```

Listing 7.3.: The algorithm that is implement in ACTS to intersect rays with AABBs. In (i) intersections are calculated, in (ii) the minima and maxima are taken, in (iii) they are combined across dimensions, and in (iv) the result is determined. These calculations are expressed in a way that is uniform across number of dimensions, and the `Eigen` library can optimize calculations.

optimization	time per intersection [ns]	fraction
none	12.8	1.0
cached inverse	7.9	0.62
full	5.8	0.45

Table 7.2.: Overview of the applied optimizations to the slab method for calculating an intersection between a ray and an AABB and their impact on measured time per intersection.

Frustum-Box intersection

For a discussion of the algorithm to calculate intersections between frustums and AABBs, it is useful to understand the way frustums were implemented in ACTS. This implementation is generic over the number of dimensions, as well as the desires number of sides. Sides, in this context, refers to the defining planes of the frustum, except for the *near* and *far* planes (figure 7.27b). Due to the specific application, the far plane is not useful and is ignored. The near plane's defining point coincides with the origin of the frustum, its normal vector coincides with the frustum direction. Keeping the near plane but placing it at the frustum origin naturally rejects intersection candidates that are *behind* the frustum. Combining the origin point, the frustum direction vector, as well as normal vectors of the sides, the frustum is fully defined. The normal vectors and direction vector then define $n_{\text{sides}} + 1$ planes in combination with the origin point. By convention, the normal vectors point toward the

volume contained by the frustum shape.

```
template <typename value_t, size_t DIM, size_t SIDES>
Frustum<value_t, DIM, SIDES>::Frustum(const VertexType& origin,
                                      const VertexType& dir,
                                      value_type opening_angle);
```

Listing 7.4.: Constructor signature of `Acts::Frustum`

The constructor of the frustum implementation, shown in listing 7.4[2], takes as input the origin point, the direction and an opening angle. It will then construct side planes according to the template parameter `SIDES`. Note that the rotation around the direction vector is undefined, and will be chosen arbitrarily during construction. This caveat is acceptable in the use case of approximating a cone with the frustum.

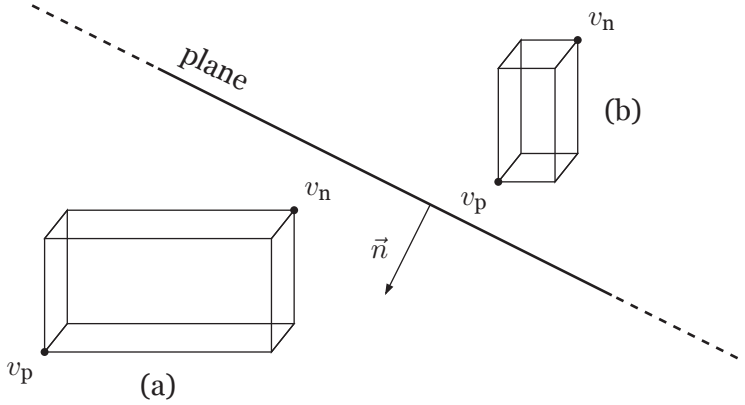


Figure 7.31.: Illustration of the calculation of the *positive* and *negative* vertices for two bounding boxes with respect to a plane. The positive vertex v_p is the one that is furthest along the direction of the plane normal $\vec{n}$.

The algorithm to calculate whether or not an AABB intersects with the frustum is based on [216]. It uses the *positive* and *negative* vertices of the bounding box. As can be seen in figure 7.31, they are two of the corner points. The figure shows two example AABBs, where v_p is the corner furthest along the direction of the normal vector $\vec{n}$, whereas v_n is the closest. It is apparent that v_p and v_n depend on the concrete plane under consideration. These positive and negative vertices can be calculated from the defining vertices of the AABB. An example for the case of the positive vertex is shown in listing 7.5.

```
auto p_vtx = (normal.array() <  0).template cast<value_type>() * fr_vmin +
             (normal.array() >= 0).template cast<value_type>() * fr_vmax;
```

Listing 7.5.: Calculation of the positive vertex for an AABB.

Here, `fr_vmin` and `fr_vmax` are the minimum and maximum vertex of an AABB relative to

[2] A full listing can be found in listing A.1

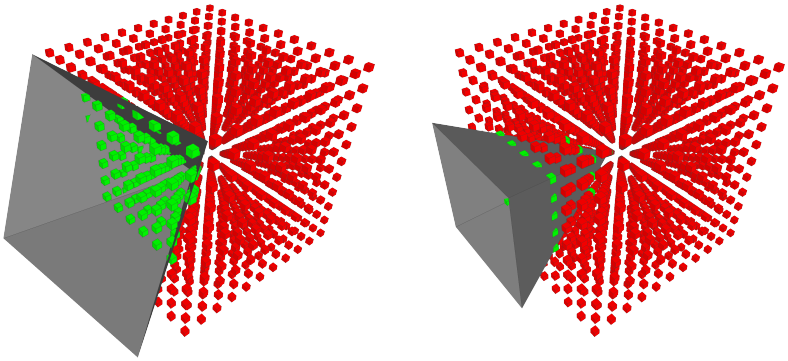
the frustum origin. In this reference frame, the positive (negative) vertex components can be determined by taking the maximum (minimum) vertex' position if the normal component is non-negative (negative). `(normal.array() < 0)` and `(normal.array() >= 0)` evaluate their conditions component-wise. The resulting boolean arrays are converted to floating point values, where `false` $\rightarrow 0.0$ and `true` $\rightarrow 1.0$. Subsequently, the components are multiplied by the minimum and maximum vertex vectors. This step effectively selects the minimum component for negative normal components, and the maximum component for non-negative components. Using this combined form avoids branching `if` statements in the calculation. By writing this operation as a single expression, and avoiding branching, `Eigen` should be able to vectorize the evaluation. This was not explicitly tested, however.

Since the navigation application is only interested in testing whether a bounding box intersects with the frustum, and does not need to determine if the box is completely contained in it, only the positive vertex is required. The check reduces to calculating whether the positive vertex is inside the frustum volume. This is done by checking whether the scalar product of the positive vertex and any of the plane normal vectors is negative, i.e. the vertex lies on the outer side of the plane. If all scalar products are positive, at least the positive vertex is contained in the frustum volume, so the bounding box has to intersect it. The full function to test the intersection can be found in listing A.2. In an isolated measurement, which tests single invocations of the intersection algorithms, the runtime of each invocation ranges between 13.5 ns and 48.8 ns. The range represents the difference between the best-case and worst-case scenarios: if the first normal check fails, the algorithm can be terminated early, conserving time. Real world performance is expected to be within this range.

Automated tests are used to make sure this algorithm yields the correct results. They are implemented by using a fixed set of bounding boxes on a regular grid. This grid of bounding boxes can be seen in figure 7.32b. A randomly selected set of frustums is constructed with three or four sides, making up a total of fourteen frustums. They are then tested against the grid of bounding boxes. To obtain a reference of which bounding boxes are intersected by which frustum, a visual method is deployed. In a first step, the grid of bounding boxes is rendered, and the color of the boxes is set depending on the intersection status. One example of such an output is exactly what is seen in figure 7.32b, where green boxes are intersected by the frustum, and red ones are not. The corresponding frustum is drawn as well. By visually inspecting this output, the correctness of the intersection results are verified. Subsequently, the bounding boxes that intersect are stored, such that when the test is run again, it can automatically check if the resulting set of intersecting boxes is identical. While the tests does not automatically determine the original correctness of the algorithm, it serves as a strong regression test. This way, changes to the implementation can be made without accidentally breaking it.

7.4.4. Bounding Volume Hierarchy navigation model

Given the implementation of algorithms to calculate intersections of rays and frustum against bounding boxes, the code is now equipped to use bounding boxes as proxies to test whether it



(a) (b)

Figure 7.32.: Visual representation of the automated tests of the frustum-box intersection logic. (a) and (b) show two different frustums that are tested against a grid of cubes. Cubes which intersect are shown in green, if they do not intersect they are shown in red.

is likely for the navigation to hit a certain object. The naive approach would be to sequentially test every eligible object's bounding box, and process with the ones for which the intersection algorithms evaluate positively. This is, however, prohibitively expensive, in particular for large numbers of objects. Simply intersecting all cell bounding boxes with a ray or frustum would take $\mathcal{O}(\text{ms})$ for the ATLAS readout calorimeter geometry. One way to solve the issue is to introduce Bounding Volume Hierarchies (BVHs). In essence, the idea is to build a tree of bounding boxes that contain other bounding boxes both logically and geometrically. The navigation can then start at the root of the tree, check whether it is intersected, and only if that is the case proceed to the children. At that point, each of the children is checked, and recursed into if they are intersected. If no intersection is found, the entire subtree is skipped.

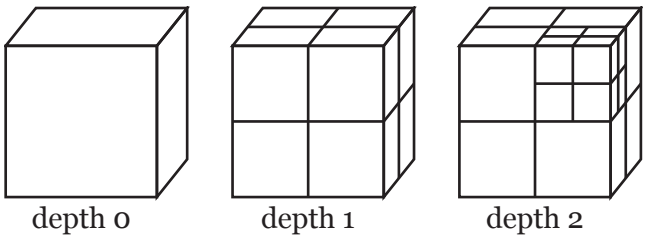


Figure 7.33.: Three levels of an octree that partitions space into eight sub-cubes at every level.

There are many different implementations of BVHs. They differ in their design based on considerations that become important depending on the application. For instance, there are different ways to partition space. Also, it is undesirable that some subtrees contain significantly more children than others. Finally, in case the underlying geometry changes over time, modifications to the hierarchy need to be efficient, and result in trees that are still acceptable in the regards stated above. For the purpose of navigation, the geometry can be considered static and thus unchanging. To keep the implementation reasonably simple, an *octree* was chosen as a BVH. This specific type of BVH, that can be seen in figure 7.33, divides

space into eight sub-cubes at every tree level for an illustration.

The octree is constructed by recursively iterating over bounding boxes and categorizing them into the eight subdivisions. Since it is crucial that at each level bounding boxes fully contain their children, the cubes are extended such that they fully contain the bounding boxes geometrically. No issue arises if during this step cubes at the same level end up overlapping with each other, since they will be checked sequentially and recursed into regardless of possible overlaps. The octree construction also accepts an envelope parameter, that increases the size of wrapping bounding boxes to ensure expected behavior at the boundaries. In addition, the routine can be configured to construct the octree only to a certain depth. Regardless of this, the construction will not add any more subdivisions if it would only wrap a single object.

A search routine that extracts navigation candidates from a constructed BVH was added to the `Acts::TrackingVolume` class. Storing the tree associations as pointers in each tree node, as well as a pointer that points at the node's neighbor, or the parent in case the node is the last child, avoids having to perform recursion. The search can be implemented using a flat loop with a *cursor* that is a pointer to the object currently being handled. The method looks for bounding boxes that are intersected and contain a cell volume, i.e. they are not only a box node in the BVH. In that case, the cell volume is decomposed into its boundary surfaces, which are collected if they are eligible as navigation candidates. Finally, the routine returns the navigation candidate surfaces sorted by their distance along a straight line from the current point in the navigation.

Modifications have been made to the main `Acts::Navigator` class that is responsible for the navigation in the ACTS propagation to call the routine from above in case a tracking volume is found that contains a BVH.

At this point, the navigator has to decide whether to use a frustum or a ray for the search in the hierarchy. This choice is based on the available information at that point, which includes the position, the momentum of the particle, and the strength of the magnetic field. The momentum and the magnetic field determine the curvature of the particle trajectory, and can be used to calculate an opening angle for a cone, or a frustum approximating a cone. Inhomogeneities in the magnetic field would complicate the calculation, so an approximately homogeneous field is assumed. As bending in the calorimetry is typically not desired, this assumption is reasonably applicable. The bending radius r can be found depending on the magnetic field $\vec{B}$ using

$$\frac{mv^2}{r} = q\vec{v} \times \vec{B}, \tag{7.17}$$

where m, q and $\vec{v}$ are the mass, charge and velocity of the particle. This relation can be modified to yield

$$r = \frac{p}{q|\hat{d} \times \vec{B}|}, \tag{7.18}$$

in which $\hat{d} = \frac{\vec{p}}{|\vec{p}|}$ is the unit direction vector of the particle at N. Equation (7.18) remains valid for relativistic particles. As the rotation around $\hat{d}$ is irrelevant later on, the calculation of the

opening angle can be handled entirely in the bending plane defined by $\hat{d} \times \vec{B}$.

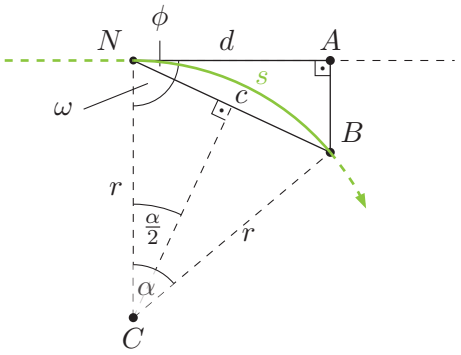


Figure 7.34.: Drawing of how the opening angle for a particle trajectory navigation query is calculated at the navigation point N.

Figure 7.34 shows how the opening angle ϕ can be calculated for a particle trajectory, indicated as a green curved line. The navigation occurs at point N. For a traversal distance s, the opening angle corresponds to the angle between the circle chord c and the original particle direction d as of point N. As $\alpha = s/r$, and NBC forms an isosceles triangle, ω can be identified as

$$\begin{aligned} \omega &= \frac{\pi}{2} - \phi \\ &= \pi - \frac{\pi}{2} - \frac{\alpha}{2} \\ \Leftrightarrow \phi &= \frac{\alpha}{2}, \end{aligned} \tag{7.19}$$

which is valid as long as $\phi < \pi$. As $\phi > \pi$ results in a search window larger than geometrically useful, this is acceptable. The calculation depends on the effectively free parameter s, which can for example be set to the maximum step length of the numerical integration.

There is, however, a caveat with this calculation of the opening angle, which is that the navigator has no way of predicting whether the magnetic field will change further along the trajectory. This means that the opening angle might not be chosen correctly to account for the full magnitude of curvature the trajectory has while crossing the volume in question. Shown in figure 7.35 is an example for two interfacing regions of homogeneous magnetic fields. A trajectory crosses the interface at point b, but the cone/frustum opening angle was estimated at point a. Due to the discrepancy between expected and actual curvature, the trajectory leaves the resolved navigation area at c.

The frustum navigation is invoked when the trajectory enters the logical volume containing the BVH hierarchy. Due to this, any significant variation of the magnetic field inside the volume will not be correctly accounted for by this opening angle calculation. To remedy this problem, the navigator would have to trigger another intersection test against the BVH as soon as it detects that the trajectory exits the cone for which the current navigation solution is valid, marked as c in figure 7.35. Since the ACTS navigator will undergo a substantial rewrite in the future, and is relatively complex to begin with, it was decided against adding

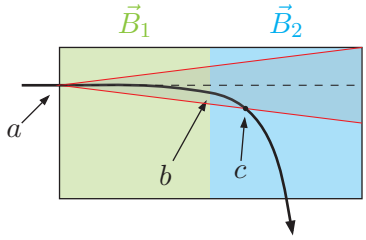


Figure 7.35.: Illustration of navigating using an approximation of the particle trajectory, that uses an envelope of the curvature. Shown are two sections with different magnetic fields. The curvature of the trajectory changes, and leaves the envelope at point c.

this re-navigation feature to the current implementation. Due to this circumstance, at the time of writing, the navigator will only choose rays to try to resolve tracking volumes that contain a BVH.

7.4.5. Performance characterization

In order to verify that the navigation model described above works, and to be able to characterize its performance, the implementation was tested against the ATLAS calorimeter readout geometry, that was introduced before. To this end, the geometry construction discussed in section 7.2 was supplemented with functionality that translates the calorimeter readout geometry. This is done by iterating over the collection of calorimeter elements provided by the *calorimeter description manager*. A special volume bounds type, `Acts::GenericCuboidVolumeBounds`, is constructed, which was added specifically for this use-case. It models the fact that the calorimeter cell volumes are not rectangular, since the calorimeter is segmented in η and ϕ, rather than cartesian coordinates. The interpretation of the defining parameters differs between the various cell categories displayed in figure 7.26. For each cell, an `AbstractVolume` is constructed, which also builds a corresponding AABB and OBB. Finally, the bounding boxes are organized in an octree, and stored in an `Acts::TrackingVolume`. The volume is then fit into the volumes wrapping the ID components, while making sure that the volume boundary surfaces match each other exactly. To achieve this, another special type of volume bounds, `Acts::CutoutCylinderVolumeBounds`, is used, that provides the necessary boundary surface slots to correctly connect to the ID volumes. A sketch of this volume type is provided in figure 7.36. The volume bounds are defined by an inner and outer half-length z_i and z_o, and inner, middle and outer radii r_{inner}, r_{middle} and r_{outer}. Also shown are the various boundary surfaces that are used to connect the outer calorimeter volume with the Pixel, SCT and TRT volumes. Special attention has to paid to the fact that the boundary surfaces need to be split correctly, such that they point to the actual volume that is located on the other side of the surface. The inner disk surfaces of the calorimeter volume from r_{inner} to r_{middle} at z_i, for instance, need to point to difference endcap volumes of the ID, depending on the radius.

With the calorimeter embedded in the ACTS tracking geometry hierarchy, it is possible to use the particle propagation to probe the navigation. Figures 7.37 and 7.38 show the steps that

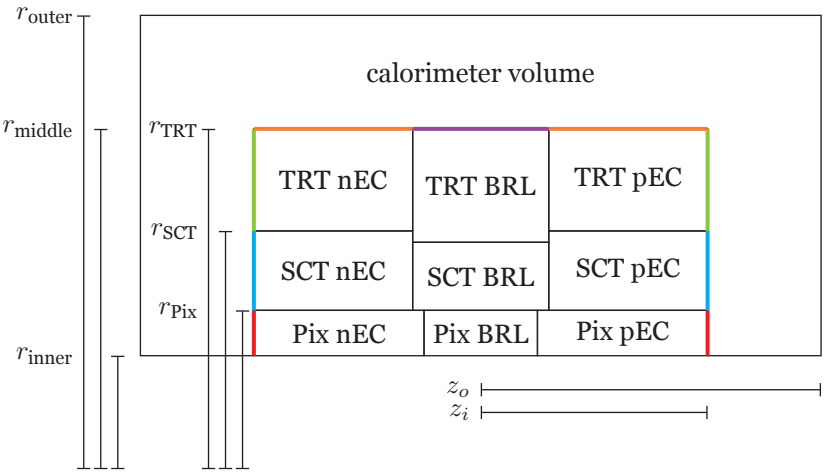


Figure 7.36.: Schematic overview over the different Barrel (BRL) and Endcap (EC) volumes used to describe the ID and the calorimeter. The calorimeter volume wraps around the ID volumes. The various boundary surfaces that connect the ID volumes to the calorimeter volume and vice versa are shown in red, blue, green, orange and purple.

the propagation makes to traverse the detector volume in blue. The steps are projected onto the rz-plane, where the z direction is on the horizontal axis. Drawn in red are propagation steps that lie on the boundary surfaces between the tracking volumes. In the central part of the plot the ID is visible. Figure 7.37 shows particles with a transverse momentum of $p_{\text{T}} = 1\,\text{GeV} - 2\,\text{GeV}$, while figure 7.38 features momenta of $p_{\text{T}} = 20\,\text{GeV} - 100\,\text{GeV}$. The particles are emitted from the exact center of the detector, the direction is randomly chosen uniformly in $\phi \in [-\pi, \pi)$ and $\eta \in [-5, 5]$. The ATLAS magnetic field is used. The majority of propagation steps are located on the entry and exit surfaces of cells. However, some intersections with boundary surfaces on the sides of cells are visible. The latter occurs more frequently at lower momenta, which is due to the fact that in these cases, the curvature of the trajectories is higher.

Comparing figure 7.37 to figure 7.38 it also becomes apparent that at lower momenta, the coverage of steps on the entry and exit surfaces reduces as well. This effect most likely demonstrates that the ray approach, which only uses straight lines, deviates too much from the actual trajectory when the curvature becomes non-negligible. This low-momentum case leads to a navigation failure, which means that the particle trajectories will not be correctly extrapolated to the end of the detector volume. In absence of a full exploitation of the frustum intersection, including the re-navigation routine outlined previously, this hypothesis cannot be fully validated.

Figure 7.39 shows selected trajectories from the same dataset of particle propagations. In Figure 7.39a the lower momentum range is visible, whereas figure 7.39b contains the higher range. While the latter shows trajectories that are approximately straight, the former clearly features more pronounced bending, due to the magnetic field, that could cause the observed navigation failures. Also shown are the approximate volumes of the ID in green, and the

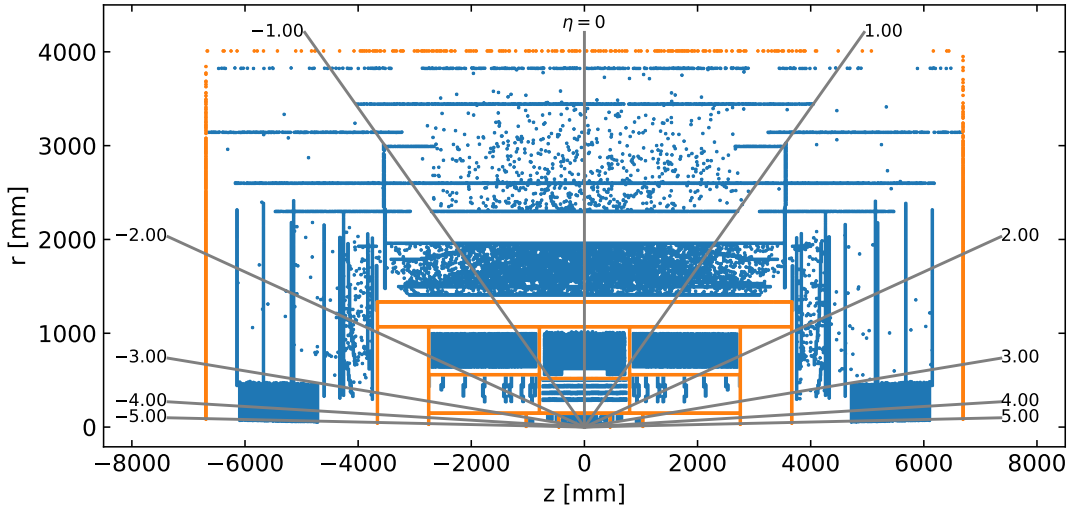


Figure 7.37.: Propagation steps of particle gun particles ranging from $p_T = 1\,\text{GeV} - 2\,\text{GeV}$ using the ACTS extrapolation through the ATLAS geometry. Blue points represent intersections with *sensitive* surfaces, whereas red points represent intersections with virtual *boundary surfaces* between logical volumes.

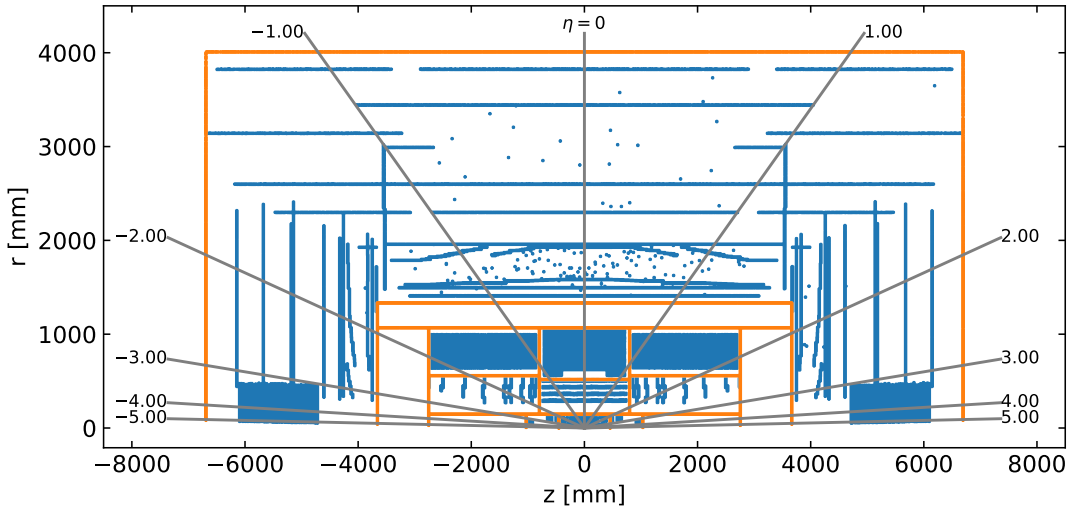
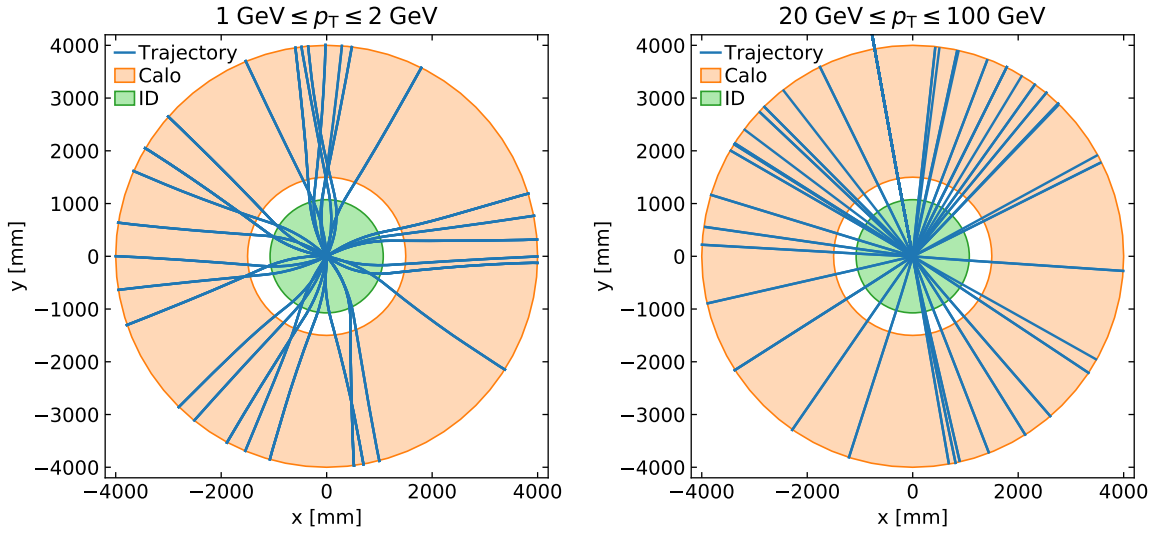


Figure 7.38.: Propagation steps of particle gun particles ranging from $p_T = 20\,\text{GeV} - 100\,\text{GeV}$ using the ACTS extrapolation through the ATLAS geometry. Blue points represent intersections with *sensitive* surfaces, whereas red points represent intersections with virtual *boundary surfaces* between logical volumes.

calorimeter in orange. In ATLAS, a strong solenoid magnet sits between these two parts of the detector. This can be seen in figure 7.39a, where the bending is strongest inside the ID volume, and the bending direction flips in the transition region between the two volumes, due to the solenoid field changing its sign.

In absence of a navigator implementation leveraging both rays and frustums to perform the navigation queries, the raw performance of the intersection calculations and the BVH

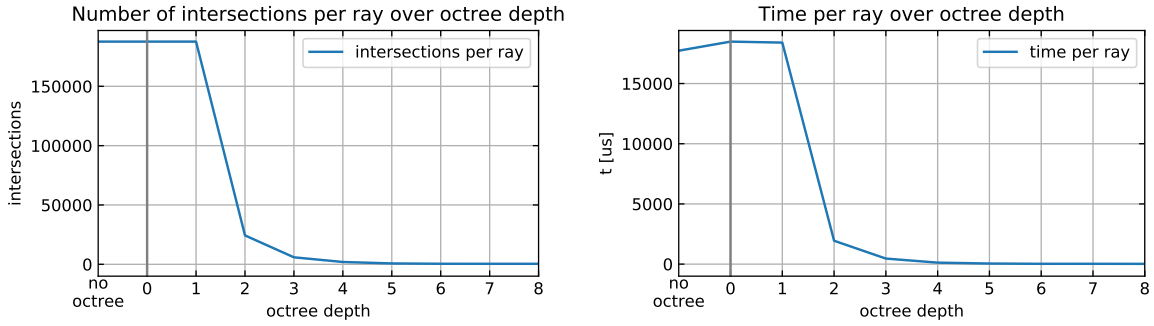


(a) (b)

Figure 7.39.: Particle trajectories passing through the ATLAS calorimeter. Propagation is calculated in ACTS. (a) shows particles with 1 GeV $\leq p_T \leq$ 2 GeV, while (b) shows particles with 20 GeV $\leq p_T \leq$ 100 GeV. It is clear that the trajectories in (a) feature much stronger curvature than the ones in (b).

traversal code was tested in an isolated setup. This has the added benefit of suppressing external effects from the Athena environment, and improving turnaround time. To this end, the numerical values of the cell parameters are written out to a ROOT file. This file is read back in by a dedicated benchmark program, that reconstructed the same geometry as is available in Athena, excluding the ID. Rather than measuring performance in combination with the numerical propagation itself, only the time needed to query the BVH for all navigation candidates is measured. To also validate that this query returns the correct candidates, a reference candidate set is collected by intersecting every single cell surface sequentially. While this is substantially slower, it is guaranteed to give the correct set. For randomly generated ray and frustum positions and directions, both the BVH search and the full search are performed, the results are compared. In parallel, the execution time of the BVH search is measured.

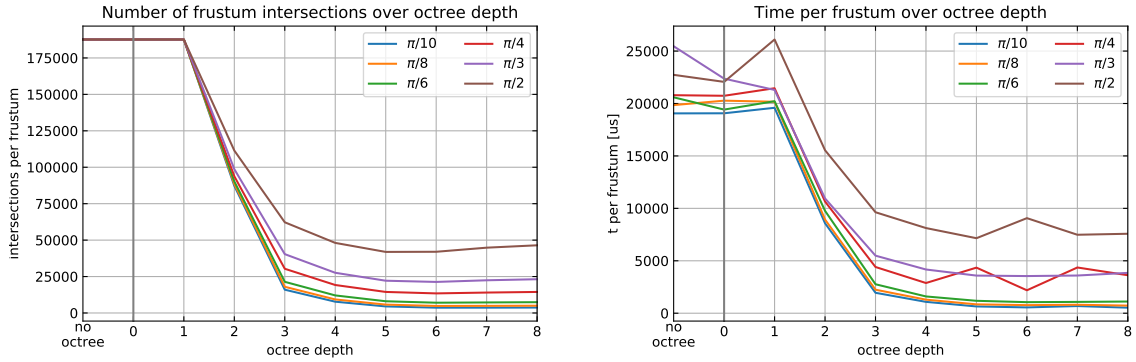
The results are shown in figure 7.40. Plotted in figure 7.40a are the number of intersections per query of a single ray against the BVH as a function of the octree depth. In figure 7.40b the measured time per intersection query is shown. The first data point is measured without the use of an octree, this is the approach used as a reference for the correctness of the algorithm described before. It also serves as the performance baseline. An octree depth of zero is practically identical to the case without any octree at all. In this case, all cells are simply wrapped inside one single bounding box. This, obviously, does not reduce the number of intersections required, and actually results in a slight increase in the required time over the trivial algorithm. Increasing the maximum depth of the constructed octree leads to a decrease in both the number of intersections required, as well as the necessary time. This



(a) (b)

Figure 7.40.: Performance characteristics of intersection tests for rays. The number of intersections per ray tested against the BVH (a) and time taken for each intersection test against the BVH (b) are shown as a function of octree depth.

decrease flattens out at an octree depth of about 3 to 4, above that, only minor decreases can be observed. The saturated values are about 500 intersections, and a time per ray of about 22 µs.

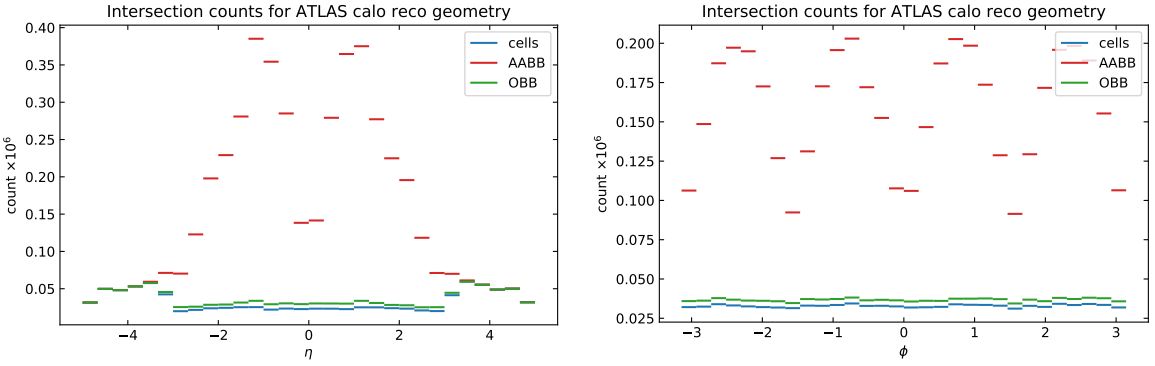


(a) (b)

Figure 7.41.: Performance characteristics of intersections tests for frustums. The number of intersections for each frustum test against the full BVH (a) and time taken for each intersection test against the BVH (b) are shown. Frustum opening angles from $\pi/10$ to $\pi/2$ are shown.

Figure 7.41 shows a very similar measurement for the intersection of a frustum against the BVH. The number of required intersections and the total time are shown here as a function of the frustum opening angle in addition to the octree depth. The general trend is the same as above: The number of intersections without an octree and up to an octree depth of one is essentially identical. The timing plot shows some variations here that are most likely due to system activity during the benchmark. A slight upward trend might be visible. Starting from an octree depth of two, both figures drop drastically, before they saturate in a similar fashion as in the case of rays. The saturation occurs at about an octree depth of three. The saturated number of intersections and time taken are dependent on the opening angle. This

observation is expected, as the number of cells that are actually in the frustum volume simply increases with the opening angle.



(a) (b)

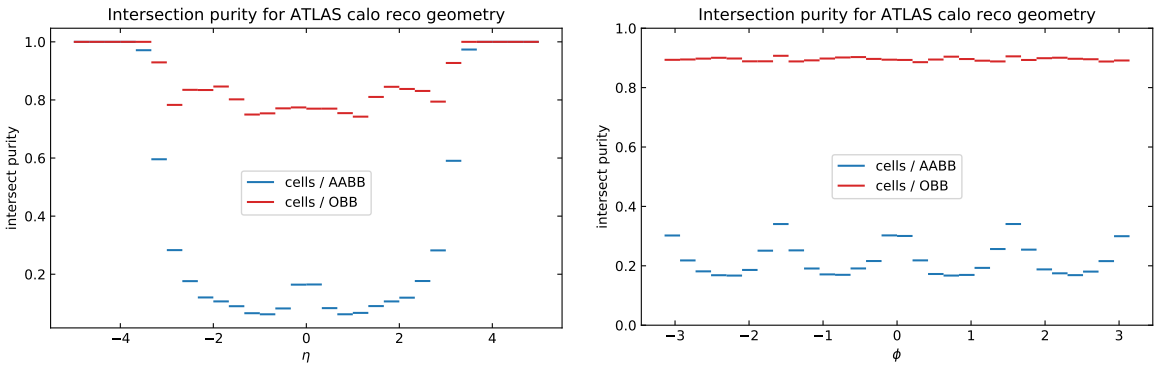
Figure 7.42.: Histograms of intersection counts for rays as a function of longitudinal (a) and transverse (b) direction. Shown are the number of required intersections with actual cells as well as the number of intersections with for AABBs and OBBs. The AABB distribution peaks at $|\eta| \approx 1$ and for a series of ϕ values. The OBB distribution remains close to the cell number.

As a next step, it is helpful to look at the number of intersections that are to be calculated for a ray in relationship to its direction. The idea is to understand for a given ray direction, how many bounding boxes report being intersected, in relation to the number of true cells that are intersected by the ray. This relationship can be interpreted as a form of *false positive* rate, where the navigation returns more cell candidates than will eventually be hit. The fraction of false positives does not affect the navigation quality in the case of a straight trajectory, as these cells will simply be rejected by the precise intersection check later on. They can however, degrade navigation performance, as more intersections have to carried out than are in fact necessary.

Figure 7.42 shows the number of intersections for a ray as a function of the pseudorapidity η and the angle in the longitudinal plane ϕ with actual cells, as well as their AABBs and OBBs. This is complemented by the corresponding ratios

$$f_{\text{AABB}} = \frac{N_{\text{cells}}}{N_{\text{AABB}}}, \qquad f_{\text{OBB}} = \frac{N_{\text{cells}}}{N_{\text{OBB}}} \tag{7.20}$$

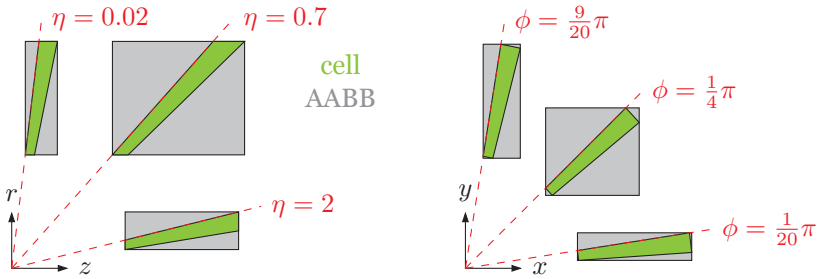
shown in figure 7.43. In η the number of true intersected cells remains flat up to around $|\eta| = 3$, before rising slightly. This can be explained by the fact that starting from $|\eta| = 3$ the rays enter the FCal volume, and encounter more cells. In this region, the number of intersected AABBs is quite comparable. This is due to the fact that the FCal cells are already cuboid shapes that are aligned with the axes, so the AABB wraps them tightly. At lower values of $|\eta|$, the number of AABB intersection rises until about $|\eta| = 1$, before dropping again. In the ratio the same behavior can be observed: f_{AABB} is close to unity in the FCal acceptance, before dropping drastically toward lower absolute values of η. A slight increase is seen again



(a) (b)

Figure 7.43.: Plots showing the ratio of cells intersected by a ray and the corresponding number of intersected AABBs and OBBs, as a function of η (a) and ϕ (b). The AABB ratio is close to one for $|\eta| > 3$ and much lower at lower values. In ϕ it features a similar peak structure as in figure 7.42b. The OBB ratio is closer to unity in all cases.

at around $|\eta| = 0$. On the other hand, in ϕ, the number of true intersected cells is flat across the entire range. The number of intersected AABBs, on the other hand, peaks at values of around $\phi = \frac{\pi}{4}, \frac{3\pi}{4}, \frac{5\pi}{4}, \frac{7\pi}{4}$. Analogously, in ϕ, f_{AABB} has minima at these values.



(a) (b)

Figure 7.44.: Illustration of cells and corresponding AABBs for varying values of η (a) and ϕ (b) in the rz- and xy-plane, respectively. See also figure 7.45 for a similar version showing OBBs.

Both of these observations can be explained by the fact that since the bounding boxes are axis aligned, cells that are oriented diagonally between axes, will be matched more poorly than cells that are more or less aligned with the axes. To illustrate this, figure 7.44 shows AABBs around a set of cell shapes. While the cell shapes themselves only qualitatively resemble the real readout calorimeter cells, the bounding boxes around them are exactly as they would be calculated in the code. As seen in figure 7.44a, for η values close to zero and $\pm\infty$, cells are aligned relatively closely with the r and z axis. Therefore, the AABB area is reasonably close to the cell area. Conversely, for intermediate values, as shown as an example for $0.7 < \eta < 0.8$, the AABB wrapping the cell has a considerably larger area than the cell itself. The same general structure exists for ϕ, as seen in figure 7.44b, where for cells around $\phi = 0, \pm\frac{\pi}{2}, -\pi$ the

areas are similar, whereas for around $\phi = \pm\frac{\pi}{4}, \pm\frac{3\pi}{4}$ there is a significant discrepancy. Both in η and in ϕ this structure can clearly be seen for AABBs in figure 7.42: the η distribution peaks at about $|\eta| = 1$, while the ϕ distribution oscillates with maxima and minima at the values mentioned before.

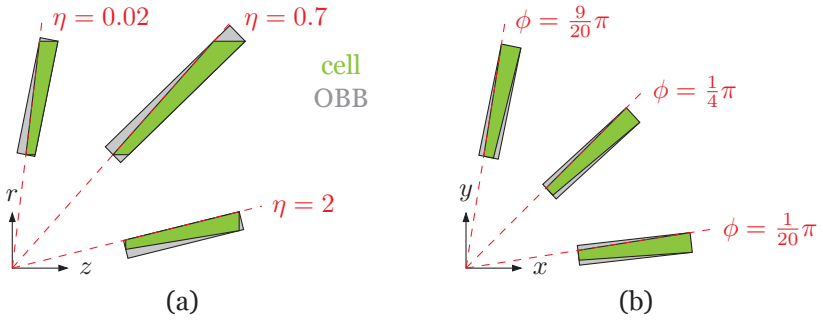
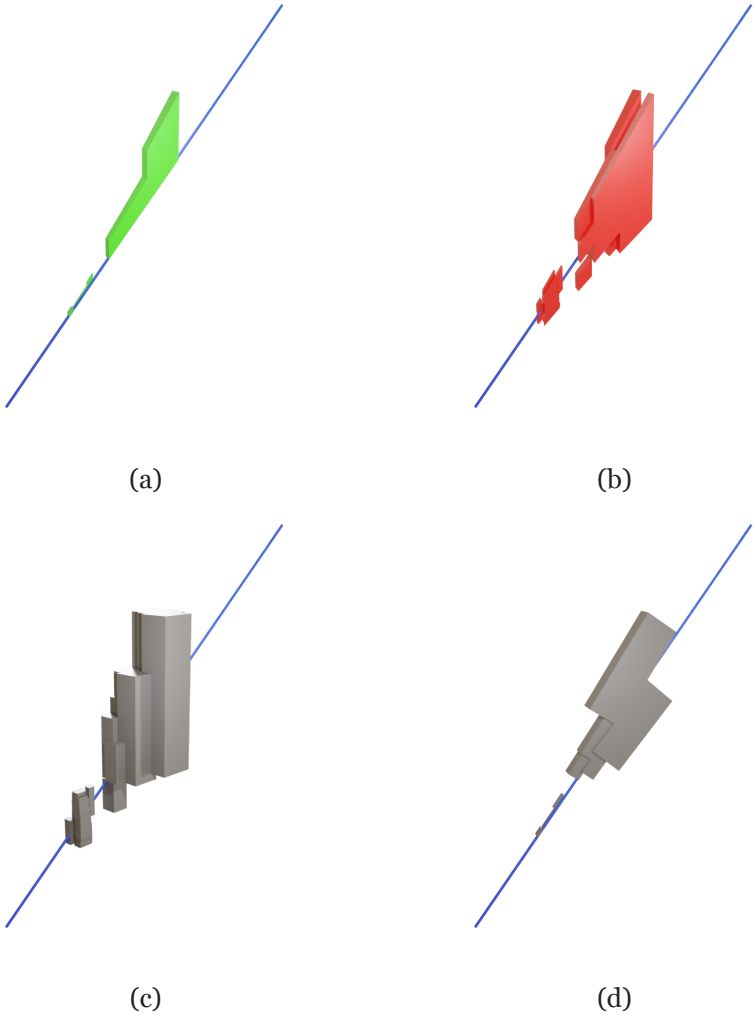


Figure 7.45.: Illustration of cells and corresponding OBBs for varying values of η (a) and ϕ (b) in the rz- and xy-plane, respectively. See also figure 7.44 for a similar version showing AABBs.

Looking at the behavior for OBBs, on the other hand, paints a different picture. Figures 7.42a and 7.43a show that the number of intersected OBBs is not drastically larger than the intersected true cells, the ratio is much closer to unity. A seen in figures 7.42b and 7.43b the same is true across the ϕ range. This observation can be understood by looking at figure 7.45, where it is visible that OBBs do not suffer the same increase in area over their wrapped cell at directions away from the axes. Again, the OBBs drawn in this figure are the ones that the code would derive, while the underlying cells are qualitative only. A certain discrepancy between the area covered by the OBB and the cell remains, however.

From the intersection counts discussed before, the conclusion is that simply using AABBs results in a large number of cell candidates that will be rejected later on. A remedy would be to use OBBs instead, which would come with a degradation of performance due to the additional required matrix multiplications. As a consequence, a combined approach was devised for the ACTS implementation: After the navigation has identified an AABB that contains a cell, it will try to intersect the cell volume's OBB. Only if this intersection check passes as well, the cell is registered as a navigation target candidate. This allows the early removal of a significant fraction of false-positive cell candidates. Again looking at figure 7.43, the resulting set of cell navigation candidates is about 20 % larger than the set of cells the ray actually intersects, but considerably lower than only using AABBs.

Finally, a three-dimensional representation of a ray query against the BVH is shown in figure 7.46. It contains pictures of the actually intersected cells (figure 7.46a), the full set of cell navigation candidates including false positives (figure 7.46b) and the AABBs (figure 7.46c) and OBBs (figure 7.46d) associated with the full set of cells.



(a) (b)

(c) (d)

Figure 7.46.: Three-dimensional representation of a ray query against the BVH for the readout calorimeter geometry. Shown are the cells actually intersected by the ray (a), all cells returned as navigation candidates, including false positives (b), as well as the AABBs and OBBs belonging to the full set of cells (c, d).

7.4.6. Summary

This section introduced a newly implemented navigation model for ACTS propagation, that can be applied to arbitrary geometries. It has fewer restrictions and is more robust with regards to the geometry.

The primary motivation for this navigation model is the navigation through the ATLAS calorimeter readout geometry, which can be of interest for various tracking applications, such as particle flow algorithms. As the readout geometry models the logical position of the readout, rather then the physical location of real hardware cells, the geometry overlaps in some places. This causes problems with the conventional navigation model implemented in ACTS. It is tailored toward layers of flat sensors, and uses a hierarchy of interconnected volumes to navigate between them. The presented alternative navigation model could also be a viable candidate for the ATLAS Muon Spectrometer, which contains many individual

sensitive volumes, spread across the overall MS volume.

After describing this conventional navigation model, and outlining its problems in the context of the calorimeter navigation, the new navigation approach was introduced. By using primitive test objects such as rays and frustums, navigation queries can be implemented in a different way. A ray can be used to approximate the trajectories of high momentum particles, as they are relatively straight even in the presence of a magnetic field. All geometry elements that intersect with the ray are potential navigation candidates. If the field is strong or the momentum too low, trajectories are curved. A frustum can be used to approximate a cone around a particle trajectory. Elements that are within the approximated cone are candidates for navigation.

Algorithms for testing whether a ray or frustum intersects with an Axis Aligned Bounding Box (AABB) were discussed, and various optimizations detailed. With the use of hierarchies of such bounding boxes, an efficient query implementation, that resolves the entire modelled geometry was shown. The algorithms were verified and shown to work correctly. Isolated performance measurements of these algorithms were performed and reported on, yielding satisfactory results.

An implementation of the ATLAS calorimeter geometry using ACTS was added to the ATLAS software. The performance of the navigation model was characterized using this geometry. Particle propagations of low and high momentum particles were studied. While high momentum particle propagation works as expected, low momenta suffer from navigation failures due to their curvature. A routine that estimates parameters for the frustum approximation of a cone to perform the navigation query was discussed. As this work is best done in the upcoming rewrite of the ACTS navigation infrastructure, this routine is not currently used, explaining the navigation failures. Runtime performance was measured in isolation and met expectations.

7.5. Design and improvement of the ACTS event data model

The following section is focused on the general design of the Event Data Model (EDM) in ACTS, as well as specific contributions made to the EDM in the context of this thesis. Previously discussed in section 6.3.2, the EDM describes how quantities that are associated with a physical event are stored, interconnected, accessed, modified and persisted. Therefore, it encompasses both how a transient memory representation of the event data works, and how it can potentially written to disk for long term storage. Different aspects have to be considered. On the one hand, the performance characteristics should be addressed. Access patterns of both reading and writing need to be taken into account, and optimized for. An object oriented EDM design lends itself relatively well to model physical objects like particles or measurements. Such a design has certain drawbacks when implemented naively, as undesirable overheads can arise. Alternative implementations should ideally be able to present interfaces that remain roughly equivalent to an actual object-oriented design.

The ACTS EDM expresses itself in a number of locations. One is the parameter represen-

tation that is used as a fundamental building block in multiple areas. Among these are the propagation, where the trajectory information is stored in instances of the parameters. At the time of writing, the parameters are modelled in a fairly standard C++ class, which uses virtual inheritance. It is related relatively closely to its predecessor in the ATLAS tracking library. Another example are measurements, which capture information on the particle trajectory obtained by sensors in the detector.

Section 7.5.1 discusses some aspects of how compile-time programming is used to improve performance of the EDM. In some cases, functionality was redeveloped using a new technique which enables additional benefits, such as time needed for compilation, and maintainability. Subsequently, section 7.5.2 reports on the implementation of a dedicated memory structure geared toward Kalman Filter (KF) and Combinatorial Kalman Filter (CKF) applications. Other currently present data structures that address a similar problems are contrasted with this new development, and its deployment in the KF and CKF discussed.

7.5.1. Compile time calculations and optimization

In modern C++, one particularly controversial feature is template-based metaprogramming. Originally designed to enable static monomorphization and thereby implement generic functions and types, it has since seen use far beyond this initial goal. As templates were discovered to form a Turing-complete sub-language, compile-time programming for all kinds of applications has become very popular. While this type of programming is extraordinarily powerful, it comes with the downsides of bloating compile times, and fiendish complexity. The latter is partially due to the template syntax being extended beyond its designed purpose. Writing non-trivial template metaprograms is therefore a striking departure from any other part of C++. This is not helped by the fact that compilers have a hard time helping the developer with error messages and diagnostics, simply because they fundamentally do not understand this embedded second order language.

Nevertheless, using metaprogramming techniques can yield significant runtime performance benefits. It eventually comes down to finding the right tradeoff between compile-time and runtime computations. One example of extensive, and perhaps excessive, use of compile-time computations is the `Eigen` library, whose application was discussed in section 6.1.3 already. It was chosen after having been picked up by the ATLAS tracking software, and replaced a wide range of other linear algebra libraries.

The rest of this section first introduces one way that the ACTS EDM uses template metaprogramming for its `Measurement` class. Subsequently, a reimplementation with favorable characteristics, that was carried out during the work for this thesis, is shown.

A discussion of the aspects of tracking EDM is best done by following the requirements of a fitting application. The basic KF fitting procedure, as introduced previously in sections 4.7 and 6.3.2, uses numeric integration to calculate a prediction of the particle trajectory on sensitive elements of the detector, iteratively. It starts form an initial estimate of the track parameters. For each encountered sensor it then converts the prediction from global to local coordinates, subsequently comparing it to the measurement.

The coordinate conversion has to be possible for different types of surfaces, such as planes, disks or cylinders. By definition, in ACTS, the coordinate frame in which a measurement on a sensor is defined can only be a strict subset of the local coordinate frame. With this assumption, for the example of a Kalman Filter, the measurement mapping function H becomes a simple projection matrix.

```
enum BoundParametersIndices : unsigned int {
  eBoundLoc0 = 0,
  eBoundLoc1 = 1,
  eBoundPhi = 2,
  eBoundTheta = 3,
  eBoundQOverP = 4,
  eBoundTime = 5,
  eBoundParametersSize, // = 6
  // various aliases
};
```

Listing 7.6.: Default local parametrization of the ACTS library. It is defined using an enumeration assigning names to parameter indices.

To accommodate this application, the library needs to be able to understand the particle trajectory parametrization. It needs to be able to convert between coordinate systems, in order to convert from the global frame, in which the numerical propagation happens, into the sensor local frames. Additionally, it needs to understand the concept of subsets of the local parametrization, in order to calculate the projection matrices that are an ingredient to the fit formulae. The backbone of the part of the ACTS EDM that supports this use case are the `ParameterSet` and `Measurement` classes. `ParameterSet` is defined on top of an enumeration that provides labels for numeric indices. These indices define the local parametrization that is used for the particle trajectory. While this is configurable by client software using ACTS, a default parametrization is shipped and can be found in listing 7.6. This corresponds to the default trajectory parameterization in ACTS:

$$\vec{x}_{\text{full}} = (l_0, l_1, \phi, \theta, q/p, t)^T \,, \tag{7.21}$$

with the local surface coordinates l_0, l_1, the azimuth and polar angles ϕ and θ, the momentum parameter q/p and the time t. In addition, various aliases are defined in the enumeration, mostly to allow referring to local coordinates with their proper names, like x/y or r/z.

The template class `ParameterSet` accepts a number of compile-time template arguments that are members of this enumeration:

```
template <ParID_t... params> class ParameterSet;
```

The size and contents of `params` can be retrieved and operated on at compile-time. `ParameterSet` ensures the given indices are sorted and unique. It provides functionality for storing values for the given parameters types in a correctly sized array, as well as a correspondingly sized

covariance matrix. When setting parameters, it makes sure that their values stay within their defined domain, if the parameter is bounded. It also knows how to construct the aforementioned required projection matrix that can be used to extract the measurement parameters $\vec{x}_{\text{sub}}$ from a dense full parameter set $\vec{x}_{\text{full}}$ like

$$\vec{x}_{\text{sub}} = \begin{pmatrix} l_0 \\ l_1 \end{pmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix} \vec{x}_{\text{full}}. \tag{7.22}$$

Here, $\vec{x}_{\text{sub}}$ corresponds to a two-dimensional location in the local surface coordinates of a sensor.

Ideally, this projection matrix would be fully constructed at compile-time. Currently, while it is possible to determine which components of the matrix need to be one and which need to be zero, it is not possible to construct the corresponding `Eigen` matrix at compile-time. Instead, the compiler emits code that will statically construct these projections matrices for all variations that were seen at compile-time as soon as the executable launches. The `Measurement` class

```
template <typename source_link_t, ParID_t... params>
class Measurement;
```

builds on top of `ParameterSet`. It stores an arbitrary subset of the full parameter set, and augments these values with a link to a corresponding measurement sensor surface. The class is used in ACTS to describe a *calibrated* measurement. Calibration, in this context, refers to the potential augmentation and correction of a measurement using the predicted intersection of a particle trajectory during a fit, as discussed in section 4.7. It also needs to contain information from which original uncalibrated measurement it was derived. As ACTS strives to be agnostic as to where the measurements originate, and in which form they are originally stored, a lightweight `SourceLink` type is used for this. It was devised and implemented as part of the EDM-related work in this thesis. This type is given as a template parameter, and a value needs to be supplied for construction. Only minimal assumptions on its interface are made. It needs to expose an `associatedSurface` method, that returns a valid sensitive surface, it needs to be copyable, and it needs to implement the equality operator. For technical reasons, it is also required to be default-constructible. Each instance of a source link is a proxy to an uncalibrated input measurement. Calibrated measurements are constructed from source links during the fitting procedure, by means of application of a calibrator helper. What exactly happens in this calibration step is opaque to ACTS by design, as it can be experiment specific.

Similar to `ParameterSet`, `Measurement` also receives a set of parameter indices at compile-time. Using the parameter value information alongside the surface link, track fitting, for example with the KF formalism, can be performed. All required calculations can be carried out using the parameter vector and associated covariance matrix. Since the parameter subset is known at compile-time, the statically assigned projection matrices can be used and shared. Apart from that, all matrix operations use compile-time fixed sizes, which are up to twice as fast as dynamic ones.

There is one problem with this approach, however, which is that the number of parameters and specific parameter subset of measurements is heterogeneous even for a single particle trajectory. Looking at the ATLAS ID, this is easy to comprehend. The Pixel measurements are defined by two parameters giving the intrinsic local cartesian position of a silicon hit. Equivalently, the SCT measurements are single-parameter silicon hits as there exists only segmentation in one spatial sensor dimension. In the case of the TRT, measurements are on straws, which effectively have a cylindrical coordinate system and are therefore also expressed as two parameters. Measurements can also have higher numbers of dimensions, for example if the sensors record timing information, or sensor measurements are grouped to capture the direction of a passing particle. This presents two challenges: measurements need to be stored homogeneously, and processing should branch into code paths optimized for a specific fixed dimensionality.

Solving these challenges requires *type-erasure* in C++, which means forgetting the concrete type in terms of its storage requirements as well as its interface. For the ACTS measurement, this is achieved by using the `std::variant` construct from the C++ STL. It implements the concept of a tagged union type. A union type is defined over a fixed number of concrete types, and allocates enough memory to be able to store the largest of the types in its block of memory. Alongside the allocated memory itself, it keeps track of which one of the concrete types is actually stored. Combining both of these pieces of information, it is possible to erase which concrete type was stored, in terms of true C++ types, but still safely cast back to that concrete type. `std::variant` does not use dynamic memory to store its value, and can be used in `std::vector`. To make use of this, however, the set of types that can possibly be stored need to be supplied to `std::variant` at compile-time. Using metaprogramming this can be achieved by calculating all possible parameter subsets[3], and supplying `std::variant` with them.

N	unique ordered subsets
1	$\{\{0\}\}$
2	$\{\{0\},\{1\},\{0,1\}\}$
3	$\{\{0\},\{1\},\{2\},\{0,1\},\{0,2\},\{1,2\},\{0,1,2\}\}$

Table 7.3.: Example of the unique ordered subsets of zero-based index packs $n \in [0, N)$ for $N \in [1, 2, 3]$. The number of subsets grows like $2^N - 1$.

The calculation begins with the enumeration from listing 7.6, which represents the full parameter set. It then needs to figure out all potential measurement subspaces that could be of interest. These do not depend on the actual meaning of any of the parameters, only on their total number. With the additional condition that the parameters in the subspaces can not be reordered, the required quantity is the set of unique ordered subsets. This set can be calculated generically for a number of parameter indices N, where there are a total of $2^N - 1$ subsets. By common convention, the lowest index is zero. Table 7.3 contains examples of

[3]This approach has since been revised in the most recent version of ACTS, and only the number of dimensions is supplied at compile time. This is sufficient to generate optimized code. Information on the unique ordered subsets is only calculated at runtime, in this approach.

these subsets for $N = [1, 2, 3]$. What is left to do is defining a variant that includes concrete instances of `Measurement`:

```
template <typename sl_t>
using FittableMeasurement = std::variant<
                              Measurement<sl_t, eBoundLoc0>,
                              Measurement<sl_t, eBoundLoc1>,
                              Measurement<sl_t, eBoundLoc0, eBoundLoc1>,
                              // other subsets ...
                            >;
```

Compile-time metaprogramming in C++ typically has ergonomics that are similar to that of functional programming languages. An example of this are lists, which need to be defined as recursive types. Iteration over them also works recursively. Programming in this way without proper syntactic support can be cumbersome, so libraries exist to alleviate this to some extent. In the original implementation of this `FittableMeasurement` type in ACTS, the `boost::mpl` metaprogramming library was used. Even with this library, a relatively large amount of deep template based logic was needed to achieve the type generation. From a technical point of view, `boost::mpl` had another drawback: since it predated language support for variadic templates, recursive preprocessor macros had to be used to implement a number of features. The recursive application of macros drives up compilation times and was identified as a sizeable factor in the overall compilation performance.

A new implementation was developed in the work for this thesis, that eschews `boost::mpl` for another library from the `Boost` family: `boost::hana`. This library fully leverages variadic templates, among other modern C++ features. As a consequence, the amount of time the compiler spends executing compile-time logic implemented in this way is much lower as is the case with `boost::mpl`. It also uses a technique that encodes types in compile-time values, which allows writing metaprogram code in a syntax that is much closer to regular C++.

Note that both implementations execute at compile time, and generate the same C++ types. The resultant `std::variant` is exactly identical, therefore yielding bit-identical output when compiled to binaries. For this reason, runtime performance of both implementations is expected to be exactly identical as well.

First, the algorithm that can generate the required unique ordered subsets needs to be understood. As an illustration, listing 7.7 shows Python code that builds the subsets at runtime. It consists of an outer loop iterating over a range from 0 to N. For each iteration, it will duplicate the already collected subsets in `comb`, while appending the iteration index. The algorithm needs to be seeded with an empty tuple to work, this invalid set is removed as a final step of the algorithm.

Since constructs such as loops are much more involved at compile-time, so whenever possible, algorithms should be implemented in terms of functional programming. The same algorithm shown in listing 7.7 can be rewritten to remove the explicit loop, and replace it with a call to `functools.reduce`. This *higher-order function* accepts another function f as its argument, alongside an iterable and an initial value. It will then execute f for each item in the

```
comb = [()]
for i in range(0, N):
  comb = comb + [c + (i,) for c in comb]
comb = comb[1:] # remove first element `()`
```

Listing 7.7.: Python snippet implementing an algorithm that calculates the set of unique ordered subsets of a set of indices at runtime. It uses an outer for loop, and an inner list comprehension. The result is found in `comb` after the execution ends.

iterable. The value returned by f is given to subsequent invocation of f. In combination, this allows *reducing* an iterable to a single value, where the concrete meaning of the reduction is up to the programmer. In other languages, reduction operations are sometimes referred to as *fold* operations. The algorithm rewritten in this way is shown in listing 7.8. The reduction function is given as a lambda as the first argument, and does effectively the same as the loop body in listing 7.7. Instead of a list comprehension, a map operation that appends the current element from the iterator to the reduction value, is employed. The initial value is also set to an empty tuple as before, which is removed after the reduction. The resulting set of subsets is exactly identical.

```
from functools import reduce
comb = [()]
comb = reduce(lambda c, i: c+list(map(lambda c_i: c_i+(i,), c)),
              range(0, N),
              comb)
comb = comb[1:] # remove first element `()`
```

Listing 7.8.: Python snippet implementing an algorithm that calculates the set of unique ordered subsets of a set of indices at runtime. It uses `functools.reduce` with a lambda containing a list comprehension. The result is found in `comb` after the execution ends.

With this functional form of the algorithm, it can be implemented in C++ and at compile-time using `boost::hana`, as shown in listing 7.9. This implementation accepts the total number of indices N at compile-time as a template parameter. It then creates the same empty tuple as shown in the Python implementations. Next, it creates a tuple from a range from 0 to N. Subsequently, it uses `hana::fold_left` to perform essentially the same reduce operation seen in the Python version. The map operation is handled by the `hana::transform`. Finally, the invalid empty tuple is removed in the same way, and the result is returned.

All that is left to do is convert this set of subsets into a set of concrete instances of the `Measurement` type. This step is shown in listing 7.10. Using another map operation, each subset is converted into a pure C++ parameter pack `auto... i` via `boost::hana::unpack`. A factory metafunction `meas_meta` is given as a template parameter, and is used to construct a concrete `Measurement` type with the expanded parameter pack. The result of this function is a compile-time tuple of measurement types. Using another unpack operation, and `boost::hana`'s `template_<T>` helper, the tuple of types can be extracted into template parameters accepted

```cpp
template <size_t N>
constexpr auto unique_ordered_subsets() {
  using namespace hana::literals; // needed for `_c` later
  // generate an empty tuple
  constexpr auto combinations = hana::make_tuple(hana::make_tuple());

  // generate range over which to fold
  constexpr auto w_range =
      hana::to_tuple(hana::make_range(0_c, hana::size_c<N>));

  // main fold expression
  constexpr auto comb2 =
      hana::fold_left(w_range, combinations, [](auto state, auto i) {
        auto mapped = hana::transform(
            state, [i](auto c_i) { return hana::append(c_i, i); });
        return hana::concat(state, mapped);
      });
  // remove the invalid empty tuple
  return hana::remove_at(comb2, 0_c);
}
```

Listing 7.9.: Implementation of the algorithm to calculate the set of unique ordered subsets of a range of values from 0 to N at compile-time, leveraging the `boost::hana` library. It accepts N as a template argument.

by `std::variant`. The metafunction `type_generator_t` then simply resolves to the populated variant type, and is aliased to `FittableMeasurement`.

```cpp
template <template <ParID_t...> class meas_meta, size_t N>
constexpr auto type_generator() {
  // generate sublists
  constexpr auto sublists = unique_ordered_subsets<N>();
  // map each sublist into a measurement
  constexpr auto measurements_h = hana::transform(sublists, [](auto s) {
    return hana::unpack(s, [](auto... i) {
      return hana::type_c<
          typename meas_meta<ParID_t(decltype(i)::value)...>::type>;
    });
  });
  // return tuple of measurements
  return measurements_h;
}

// invoke the metafunction `type_generator` and extract the returned type
template <template <ParID_t...> class meas_meta, size_t N>
using type_generator_t = typename decltype(hana::unpack(
    type_generator<meas_meta, N>(), hana::template_<std::variant>))::type;
```

Listing 7.10.: Implementation of two metafunctions. The first constructs a tuple of concrete `Measurement` type instances from a calculated set of unique ordered subsets. The second unpacks the tuple of types into a single `std::variant`, that can be used to type-erase concrete measurements.

By removing the implementation based on `boost::mpl`, the compile performance is im-

proved by about 10 % in both memory usage and duration, and the resulting code is much more concise and readable than before. At the same time, the binary output of both implementations is exactly identical. As the previous version had to use many conventionally written recursive metaprogramming constructs its maintainability is trumped by the `boost::hana` implementation. The latter almost reads like *normal* C++ code, with readable function names and an algorithm flow that can be followed more easily.

7.5.2. Column-based storage model for filtering applications

Fitting algorithms, such as the Kalman Filter, often operate on an already defined sequence of measurements that are assumed to belong to the same trajectory. Alternatively, a mechanism to dynamically find compatible measurements based on the best estimate for the particle trajectory up to that point is used. This latter approach, often called Combinatorial Kalman Filter (CKF), is similar to a *single* KF fit from a mathematical point of view, but differs in the algorithmic structure. Starting from a trajectory seed, the CKF propagates a track state along the particle trajectory. Whenever it encounters a sensor, it will look for measurements. If more than one measurement is found, it will select a subset of them based on quality criteria and branch out. Figure 7.47 sketches the working model of a CKF. The first sensor only contains a single measurement. On the second sensor that is encountered, two measurements are found. The CKF traces both trajectory candidates. A selection of the best trajectory is typically done at a later stage.

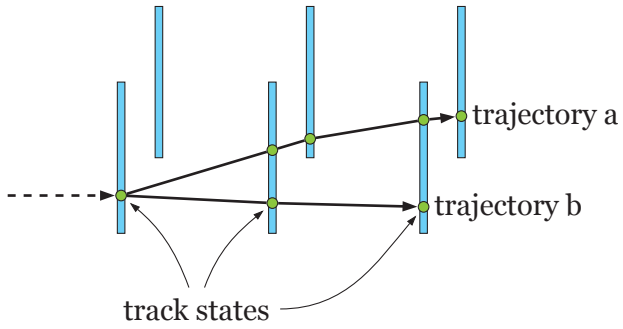


Figure 7.47.: Illustration of a trajectory branching into two trajectories across a set of six sensors. One sensor has more than one measurement, the others only have a single measurement, in this example.

Both variations of such a filter need to interact with the EDM, in order to locate measurements, potentially calibrate them, and store intermediate *track states* which can contain multiple sets of trajectory parameters. Additionally, track states are also used to store trajectory intersections where a measurement is expected, but none was found, or for material surfaces. In case the KF is supposed to perform a smoothing step to extract the best estimate of the particle trajectory, a set of predicted, filtered and smoothed track parameters needs to be stored.

The conceptually simplest approach to this is to create a class that can hold each of the required components. In ACTS, such a class is present in the form of `TrackState`, and is

```
template <typename source_link_t, typename parameters_t>
class TrackState {
public:
  /// ... interface typedefs
  /// ... various constructors, destructor, assignment and copy operators

  const Surface& referenceSurface() const;
  void setType(const TrackStateFlag& flag, bool status = true);
  bool isType(const TrackStateFlag& flag) const;
  TrackStateType type() const;
  std::optional<size_t> size() const;

  struct {
    std::optional<Parameters> predicted{std::nullopt};
    std::optional<Parameters> filtered{std::nullopt};
    std::optional<Parameters> smoothed{std::nullopt};
    std::optional<Jacobian> jacobian{std::nullopt};
    double pathLength = 0.;
    double chi2 = 0;
  } parameter;

  struct {
    std::optional<SourceLink> uncalibrated{std::nullopt};
    std::optional<FittableMeasurement<SourceLink>> calibrated{std::nullopt};
  } measurement;
};
```

Listing 7.11.: Interface of the `TrackState` class that can be used to capture required components of an intermediate state during fitting. Shown are the template parameters, several methods for getting and setting properties, as well as two nested structures that contain components relating to the trajectory parameters, and a possible measurement.

shown in listing 7.11. Individual components of the track state can be set one after the other, and some might not be set at all depending on the application. Therefore, the track state class uses `std::optional` to capture components that might not be set. `std::optional` internally allocates memory in-place for the contained value, thus not needing a dynamic allocation. It also stores whether or not the value is set. A downside is that the type has the full memory footprint of the contained type, even if no value is set. As memory consumption of track reconstruction needs to be managed in order to remain inside current and future memory budgets, this factor should not be neglected.

The `TrackState` class uses a nested struct for the components related to the track properties, such as the three parameter sets, the accumulated jacobian matrix relative to the preceding track state, the total path length and a χ^2 value. The grouping serves the purpose of making sure that for each set of parameters, the same data can be stored. On the other hand, components pertaining to the measurement of the state are captured in another nested struct. The track state directly stores a `SourceLink` for a possible uncalibrated measurement, while using the `FittableMeasurement` type that was discussed in section 7.5.1 for the calibrated measurement.

The class also includes several methods to inspect the track state, such as an enumeration

used as a flag to categorize states, accessing which sensitive surface a track state is associated with, and the size of a possibly contained measurement.

One consequence of using a track state class as was just shown is how its data is organized memory. In C++, the members of a structure are layed out sequentially, meaning for example that

```
struct V { double x; double y; double z; };
```

will result in consecutive[4] blocks of memory with `sizeof(double)` bytes each. Suppose, the `struct V` from above is stored in an `std::vector`. The resulting memory layout will be packed such that each instance's memory is followed by the next one's. This layout is shown in figure 7.48, and is typically referred to as Array of Structs (AoS).

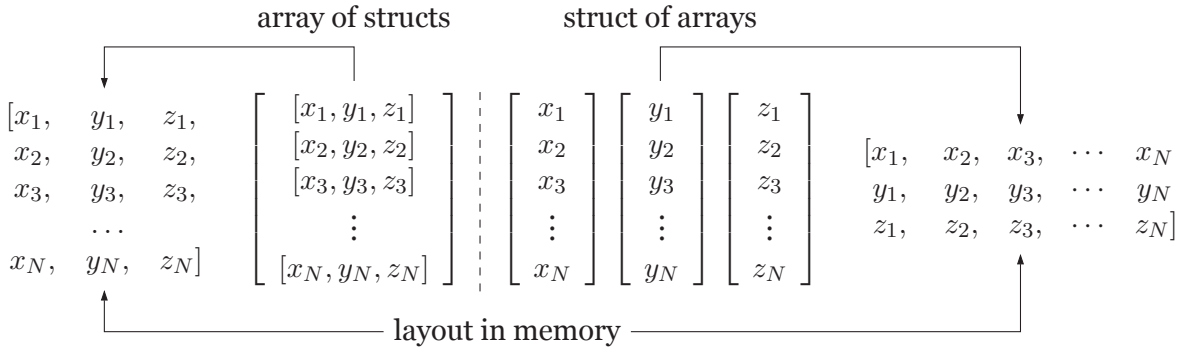


Figure 7.48.: Schematic overview of the Array of Structs and the Struct of Arrays approach. The left side shows how a sequence of structures logically appears to the user, and how it is layed out in memory. The right side shows how the same information can be stored such that each component uses its own sequence. The resulting memory layout is also shown.

Another approach to store the same information is the Struct of Arrays (SoA) approach. Instead of defining a single structure that gets repeated multiple times, each property of the structure is separated out into its own sequence. In the example of `struct V` from before this would mean having one sequence for `double x`, `double y` and `double z` each. Such a layout is visible on the right half of figure 7.48. Here, the information is spread across three separate sequences, the layout in memory is such that all values of `double x` are stored contiguously, followed by all the values for `double y` and `double z`, respectively. The stored information is exactly identical, the memory layout and the logical presentation is different, however.

Depending on the access pattern, both approaches may be more suitable from a raw memory performance point of view. In case all members of the stored structure are accessed when iterating over the sequence, SoA results in better cache locality. This is due to the fact that CPUs generally read more than strictly needed when accessing memory. The additional data that was read is stored in dedicated cache memory that is very fast. Subsequent reads to access memory locations that are already cached by this mechanism occur much faster. If, on

[4]This only strictly true if no padding is applied.

the other hand, operations operate in a column-wise fashion, the cache is more efficient if the data is arranged in an AoS.

This difference becomes even more pronounced when using Single Instruction Multiple Data (SIMD), which is a category of special CPU instructions that work in tandem with special vector registers. In essence, rather than loading and storing individual data points in registers, and executing instructions on them one by one, vector registers are filled with a fixed number of data points. Special vector instructions can then be executed on the entire register at the same time. Used correctly, this technique can yield large performance benefits.

To support the development of a CKF, and to evaluate the viability of using a column-based SoA like storage solution, an implementation of this concept was added to ACTS as part of the work for this thesis. As briefly mentioned before, the track state structure of a CKF resembles a tree, since the CKF explores multiple available measurements, and branches out. This tree-like structure is foreseen in the design of the underlying EDM. The public interface of this infrastructure consists of a class `MultiTrajectory`, as well as a track state proxy object. This proxy object is meant to provide an AoS like view into the SoA `MultiTrajectory`.

The backbone of the implementation is a structure that represents individual columns of data being stored. While for some of the columns a simple `std::vector` is sufficient, other columns use a custom type. This is due these columns representing mathematical matrices and vectors. A type from the `Eigen` library is used to allow the definition of a column that has a dynamic number of rows, but a fixed number of columns. Vectors can trivially be encoded in this higher-level matrix by writing them to individual rows. Matrices on the other hand are serialized such that their contents also fit in a single row. Since the number of rows and columns of these inner matrices are known and fixed, this conversion can occur automatically. The resulting collection of columns is therefore not fully SoA, since the individual elements of the vectors and matrices are still stored contiguously, but they are still grouped by their category (e.g. predicted, filtered, smoothed parameters). As is the case with `std::vector`, this type called `GrowableColumn` allocates a certain block of memory, and keeps track of how many elements are actually stored in it. If this number exceeds the available capacity, an additional allocation is performed, and the existing elements are copied to the newly allocated block of memory.

Each track state occupies a set of indices across the various columns that are stored. To keep track of which column's specific index belongs to which track state, an additional sequence of index tuples is maintained. The definition of the index tuple can be found in listing 7.12. 16 bit integers are used as the index numbers, and the maximum value is used as a special tag value to indicate an *invalid* index. Having this tag value allows marking a specific component of the index tuple to be interpreted as not being set. In that case, no storage is allocated in the corresponding column, and memory can be conserved.

The details of how the vectors and matrices are stored are hidden by the interface: public getters of the track state proxy return proxy types that behave like regular `Eigen` vectors and matrices. They keep track of which row of the column storage they originated from. This concept is illustrated in figure 7.49. It shows an instance of `MultiTrajectory`, where only a

```
struct IndexData {
  using IndexType = uint16_t;

  static constexpr IndexType kInvalid = UINT16_MAX;

  IndexType irefsurface = kInvalid;
  IndexType iprevious = kInvalid;
  IndexType ipredicted = kInvalid;
  IndexType ifiltered = kInvalid;
  IndexType ismoothed = kInvalid;
  IndexType ijacobian = kInvalid;
  IndexType iprojector = kInvalid;

  double chi2;
  double pathLength;
  TrackStateType typeFlags;

  IndexType iuncalibrated = kInvalid;
  IndexType icalibrated = kInvalid;
  IndexType icalibratedsourcelink = kInvalid;
  IndexType measdim = 0;
};
```

Listing 7.12.: Definition of the index tuple that is used to keep track which track state in a `MultiTrajectory` is associated with which indices across the various columns. The underlying type is a 16 bit integer, and its maximum value is used as a tag to signify an index being unset.

select number of columns are visible, alongside the track state index. The index is used to resolve the additional index tuple that contains detailed information on where in the individual columns the track state's data is stored. On the right an instance of `TrackStateProxy` is displayed, that contains a link back to the track state at $i = 2$. It exposes methods like `index()`, `predicted()` and others, which directly expose the data blocks stored in the underlying storage.

Another aspect of using this index based association from track states to the columns is that it also allows for multiple track states to share storage for some components, but have their own storage for others. An illustration of this is shown in figure 7.50. Consider again the application example of a CKF. If the algorithm encounters a sensor with multiple measurements it branches out, and thus needs to create one track state per measurement found. However, the predicted trajectory and parameters and associated covariance are identical. This duplication can be avoided by creating an index tuple for the first track state, and then reusing the indices for the predicted parameters and covariance in any subsequently created track states. The example in figure 7.50 shows exactly this: only one set of predicted parameters and covariance is pointed at by both track state proxies. However, individual filtered parameters and covariances are available for them.

Finally, each track state also contains a member `iprevious` of type `IndexType`. This is the index of the index tuple which makes up the preceding track state in a sequence. Using this association field allows the creation of the required tree of track states, rather than being restricted to a single sequence. If `iprevious` is set to `kInvalid`, the track state is a root of the

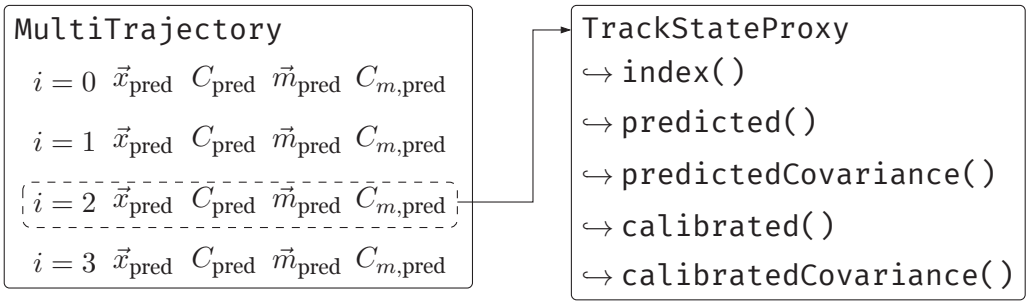


Figure 7.49.: Schematic representation of the `MultiTrajectory` data structure. On the left a selected number of columns available is shown, along with the track state index. On the right, the track state proxy illustrates how it is possible to access an individual row by using the index, and a link back to the `MultiTrajectory`.

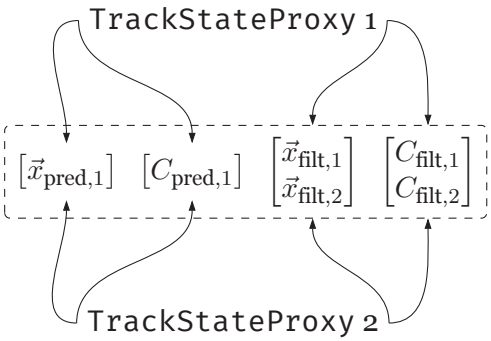


Figure 7.50.: Illustration of multiple track state proxies sharing some of their components. In this case, both proxies share the predicted parameters and covariance, while retaining separate storage for filtered parameters and covariance.

tree, meaning it does not have a preceding track state.

The public interface of `MultiTrajectory` is shown in listing 7.13. The class is templated on the source link type to allow experiment specific customization. It features two methods to create additional track states. The first one is used to create a track state in the `MultiTrajectory` from an instance of the separate `TrackState` class (as seen in listing 7.11). The second one adds a track state, but all components are set to contain zeroes, rather than specific values. Using a bit mask `TrackStatePropMask` (see listing 7.14), it is possible to steer which components are copied from the source track state, or allocated if an empty track state is added, respectively.

The methods to add track states return the index of the newly added one. Using the `getTrackState` methods allows retrieving a proxy for a given index. The method exists twice, one returns `ConstTrackStateProxy`, the other returns `TrackStateProxy`. This is to ensure that if client code has a constant reference of the `MultiTrajectory`, retrieving a proxy does not allow it to perform mutations on the data storage. The two proxy classes enable returning both mutable and immutable proxies.

Finally, the class contains two methods called `visitBackwards` and `applyBackwards`. Both of these are helpers to iterate a single sequence of track states. It starts from an index given as an argument, and will successively unpack the index of the preceding track state, until it

```cpp
template <typename SourceLink>
class MultiTrajectory {
public:
  template <typename parameters_t>
  size_t addTrackState(const TrackState<SourceLink, parameters_t>& ts,
                       TrackStatePropMask mask = TrackStatePropMask::All,
                       size_t iprevious = SIZE_MAX);

  size_t addTrackState(TrackStatePropMask mask = TrackStatePropMask::All,
                       size_t iprevious = SIZE_MAX);

  ConstTrackStateProxy getTrackState(size_t istate) const;

  TrackStateProxy getTrackState(size_t istate);

  template <typename F>
  void visitBackwards(size_t iendpoint, F&& callable) const;

  template <typename F>
  void applyBackwards(size_t iendpoint, F&& callable);

  // ...
};
```

Listing 7.13.: Excerpt of the public interface of `MultiTrajectory`. It is templated on the source link type. Methods exist to add a track state from a `TrackState` object, or create an empty one. Also provided are getters that retrieve proxy views for track states. Finally, it contains convenience methods to traverse a series of track states.

arrives at a root track state. At that point, the iteration stops. At every step, a callable function, that can be provided as an argument to the function is called. The difference between the two methods is that only the *apply* version is allowed to mutate the track states, and is therefore passed a mutable track state proxy. On the other hand, the *visit* version can only be used to inspect the track states, for example to count the number of holes or outliers in a trajectory.

```cpp
enum struct TrackStatePropMask : uint8_t {
  None = 0,
  Predicted = 1 << 0,
  Filtered = 1 << 1,
  Smoothed = 1 << 2,
  Jacobian = 1 << 3,

  Uncalibrated = 1 << 4,
  Calibrated = 1 << 5,

  All = std::numeric_limits<uint8_t>::max()
};
```

Listing 7.14.: Bit mask to steer component allocation for a track state. It is implemented as a strong enumeration type, to prevent automatic conversion. All conventional operators for bitwise operations are available and can be used as expected. A special `All` value is also available, that enables all possible components.

As seen in listing 7.15, which shows the declaration and member variables of the track state proxy class, only a pointer to the `MultiTrajectory` instance from which it originated is stored, along with the index of the track state. All access methods use this information to navigate to the index tuple, and from that to the elements of the various columns in the storage. It explicitly receives the trajectory parametrization size and the maximum measurement size as template parameters, in order to correctly define the parameters vectors and covariance matrix dimensions needed. Both numbers are combined to define the projection matrix dimension. Finally, a compile-time boolean is used to implement the mutability behavior mentioned before. Depending on this boolean value some methods are removed from overload resolution, therefore conditionally enforcing immutability.

```
template <typename source_link_t,
          size_t N,
          size_t M,
          bool ReadOnly = true>
class TrackStateProxy {
  private:
  TrackStateProxy(ConstIf<MultiTrajectory<SourceLink>, ReadOnly>& trajectory,
                  size_t istate);
  ConstIf<MultiTrajectory<SourceLink>, ReadOnly>* m_traj;
  size_t m_istate;
};
```

Listing 7.15.: Declaration of the track state proxy class, showing the template parameters as well as the private data members. Only a pointer to the owning data storage and an index are stored. Both are used for any data retrieval the proxy performs. `N` and `M` define the maximum number of parameter and measurement dimensions. `ReadOnly` marks the proxy object as immutable if set to `true`.

Access to the components of the track states is possible via methods like

```
using Parameters = Eigen::Map<ConstIf<Coefficients, ReadOnly>>;
using Covariance = Eigen::Map<ConstIf<Covariance, ReadOnly>>;

bool hasPredicted() const;
Parameters predicted() const;
Covariance predictedCovariance() const;
```

for the predicted track parameters. `Parameters` and `Covariance` are `Eigen` map types. These map types provide read and write access, depending on the value of `ReadOnly`, and behave like regular vectors and matrices. `hasPredicted` indicates whether or not this vector and matrix are allocated in the backing storage for this track state. An additional method

```
BoundParameters predictedParameters(const Acts::GeometryContext& gctx) const;
```

that constructs an instance of `BoundParameters`, which is used in other parts of the software, is also available. The same three methods are also available for filtered and smoothed parameters

and covariances, respectively. Aside from these parameters, the most important component is the measurement. The track state proxy exposes the uncalibrated measurement as a source link via

```
bool hasUncalibrated() const;
const SourceLink& uncalibrated() const;
template <bool RO = ReadOnly, typename = std::enable_if_t<!RO>>
SourceLink& uncalibrated();
```

Here, the last method provides mutable access only if `ReadOnly` is equal to `false`. On the other hand, the different properties of the calibrated measurement can be accessed as well using:

```
bool hasCalibrated() const;
const SourceLink& calibratedSourceLink() const;
template <bool RO = ReadOnly, typename = std::enable_if_t<!RO>>
SourceLink& calibratedSourceLink();
Measurement calibrated() const;
MeasurementCovariance calibratedCovariance() const;
```

In this case, access to the calibrated source link is possible, aside from access to the calibrated measurement parameters and covariance. `Measurement` and `MeasurementCovariance` are the same map types that are used for the parameters above. Again, mutable accessors are only available if the track state proxy does not carry the `ReadOnly` flag.

Access to the jacobian and the projection matrix, which are needed during filtering and smoothing, is possible with the following methods, again using the same protections for immutability:

```
Covariance jacobian() const;
bool hasJacobian() const;
Projector projector() const;
bool hasProjector() const;
template <typename Derived, bool RO = ReadOnly,
          typename = std::enable_if_t<!RO>>
void setProjector(const Eigen::MatrixBase<Derived>& projector);
```

As opposed to all the other matrices, the projection matrix uses a dedicated type

```
using Projector = Eigen::Matrix<typename Covariance::Scalar, M, N,
                                Eigen::RowMajor | Eigen::AutoAlign>;
```

since it is stored in a different way. Rather than using a dense matrix for every projector, the components of the matrix are stored in a bitset. Since all elements of the matrix are either 0 or 1, this allows a much more compact representation in only 36 bit rather than 2304 bit

when using double precision floating point values in case of a 6×6 matrix. Due to the fact that the projection matrix has to be unpacked from the bitset, the `projector` method returns the matrix by value, rather than providing a view into the storage or a reference. Also for this reason, an additional setter is provided. While the bitset representation yields excellently compact storage, packing and unpacking a conventional matrix to and from the bitset is a potential performance consideration that needs to be studied further. One potential remedy would be to statically construct all projector combinations, and use the bitset to index into this collection of projectors.

Aside from the accessors shown so far, the class also provides similar accessors for the reference surface, the χ^2 value, the accumulated path length, `TrackStateType` and the measurement dimension.

$$\begin{bmatrix} x_{00} & x_{10} & x_{20} & 0 & 0 & 0 \\ x_{01} & x_{11} & x_{21} & 0 & 0 & 0 \\ x_{02} & x_{12} & x_{22} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Figure 7.51.: Illustration of a filled 3×3 submatrix in the top left corner of an *overallocated* 6×6 matrix.

It is critical to outline the way the measurement vectors and matrices are stored and used. The data structure is supplied with the maximum measurement dimension. As a consequence, all vectors and matrices are overallocated. If the dimensionality of a measurement is lower than the maximum, the rest of the full-size matrix will still be present, as shown in figure 7.51. Here the top left 3×3 submatrix is filled, while the rest of the overall 6×6 matrix is equal to zero.

The track state proxy provides two ways to navigate from the full-sized vector to the correctly sized one for the underlying measurement. It uses the dimension that is stored in the index tuple (see listing 7.12) to determine what the actual size should be, with the assumption that submatrices are always aligned with the upper left of the full matrix. Calls to two dedicated functions

```
auto effectiveCalibrated() const;
auto effectiveCalibratedCovariance() const;
```

will allocate dynamically sized views into the underlying storage and return them. This is helpful mostly for debugging, but lacks the desireable performance characteristics of statically sized operations. The function

```
EffectiveProjector effectiveProjector() const;
```

behaves similarly, but returns a dynamically sized copy, since the matrix is stored as a bitset rather then a full matrix. Dynamically sized matrices and vectors of this type can be avoided by using a helper function

```
template <typename L, typename A, typename B>
auto visit_measurement(A&& param, B&& cov, size_t dim, L&& lambda);
```

that will generate code for each possible dimension at compile-time, and dispatch into the correct branch at runtime based on the provided dimension. It accepts a callable function that will be executed with the correctly and statically sized parameter vector and covariance matrix of the measurement. The dispatch mechanism is hidden from both the caller and the callable.

The existing KF implementation based on the `TrackState` class seen in listing 7.11 was adapted to use the new `MultiTrajectory` storage backend. Most of the access patterns could be replicated without major changes in the structure and logic of the code. The mathematical operations for the filtering and smoothing were adapted by simply switching out the accessor. This was possible due to the map types, that provide readable and writeable views into the data structure. The runtime CPU performance of the adapted KF was studied and compared to the original implementation. No significant performance impact was found, which is consistent with the notion that the access pattern of the KF, which operates on a single sequence of measurements, does not specifically benefit from an SoA data structure. Since no performance degradation occurs, switching the KF to the new data structure is acceptable, and has the added benefit of sharing the storage implementation with the CKF. The `MultiTrajectory` was successfully used in the implementation of a CKF that has since been added to ACTS by another contributor.

7.6. Memory management of shared objects

In both the ATLAS tracking library and the ACTS framework, the `Surface` class is the fundamental building block for the majority of the geometry description. It provides the local coordinate frame and parametrization, and contains information on the shape of the underlying sensitive element. Additionally, it also contains a link to an optional experiment-aware detector element object, that can be used to access domain specific functionality.

Most commonly, surfaces are created during tracking geometry construction, and subsequently owned by the tracking geometry itself. This makes sense, since the sensitive elements are fixed during the lifetime of the tracking geometry, and do not need to be replaced in between. In some scenarios, however, surfaces are created transiently. Examples of this can be found in track fitting, for example using a KF, where sometimes additional pseudo-measurements need to be added to stabilize the fit. Also in this context, when trying to perform a combined fit of two separate but matching tracks, this problem arises. Here, the initial parameter of the second track need to be associated to a surface, to provide the correct

mathematical environment for the KF. In such a case, typically a *curvilinear* surface is used, whose normal is the direction vector of the particle trajectory. The local coordinates l_0 and l_1 can be set to zero, and their directions are arbitrary, as the rotation of the local coordinate system around the normal vector is not fixed. Such temporary surfaces need to be created and deleted correctly, to ensure safe memory behavior.

To understand the issue, it is convenient to discuss briefly the way heap allocation in C++ is most commonly done. Suppose a structure `struct A`. Memory allocation on the heap can be achieved as shown in listing 7.16. In `funcA`, `new A` performs a heap allocation and `delete a` at the end of the function deallocates. This pattern works, but is dangerous: if the function `doSomething` throws an exception, the instance at `a` is never deallocated. In `funcB`, which simply omits the `delete`, this is even more obvious. Both of these example constitute memory leaks, which means memory is allocated and never released. If such code is called frequently, it can fill up memory space quickly, and cause problems.

```
void funcA() {
  A* a = new A(); // allocation
  doSomething(a); // usage
  delete a; // deletion
}
void funcB() {
  A* a = new A(); // allocation
  doSomething(a);
  // no deletion!
}
```

Listing 7.16.: Examples of manual heap memory allocation and deletion. The first function manually allocates, passes a pointer to another function and deletes after. The second function omits the deletion, therefore presenting a memory leak.

In the ATLAS, and initially in ACTS as well, this method is used to allocate surfaces. The tracking geometry implicitly assumes ownership of the surfaces it knows about. At the end of its lifetime, it will iterate over all the surfaces and explicitly deallocate them. This is generally safe, since the destructor is guaranteed to be called, even in the event of exceptions. To enable temporary surfaces, the surface class contains a boolean member `isFree`, which is `true` if the surface is not owned by the tracking geometry. Code which stores surfaces always needs to call the method to determine if the surface is free. If so, it will clone the surface, therefore storing an instance which only belongs to that particular piece of code. If it is not, it will use the pointer as is. Such code then also needs to ensure that, upon deletion, the surface is deleted if, and only if, the surface is free, so if the clone operation has taken place initially. This entire set up is confusing and potentially dangerous, since ownership needs to be implemented in every user of the surface class. The frequent cloning operations can also potentially be an unnecessary overhead. Listing 7.17 shows an example of this explicit pattern from the class `Trk::CaloCluster_OnTrack` of the ATLAS software[5]. A ternary operator is used to check for a free surface in the constructor, and an `if` is used to guard the `delete` in the destructor.

[5]Tracking/TrkEvent/TrkCaloCluster_OnTrack/src/CaloCluster_OnTrack.cxx

This pattern becomes even more problematic when copies of objects containing pointers to surfaces are made, or assignments are performed. For each of these operations, the ownership status of the surface needs to be explicitly assessed, and clones have to be created if necessary.

```
Trk::CaloCluster_OnTrack::CaloCluster_OnTrack(
  const Trk::LocalParameters& locpars,
  const Amg::MatrixX& locerr,
  const Trk::Surface& surface,
  const Trk::EnergyLoss* eloss)
: MeasurementBase(locpars, locerr) {
  m_surface = surface.isFree() ? surface.clone() : &surface;
  // ...
}

Trk::CaloCluster_OnTrack::~CaloCluster_OnTrack() {
  if (m_surface && m_surface->isFree()) {delete m_surface;}
  // ...
}
```

Listing 7.17.: Example of the way free surfaces are handled in Athena. The constructor checks if the function is free, and if so clones it. The destructor will delete the surface if it is free, to recover unused memory.

Since C++11, the problem discussed before can be solved by using *smart pointers*. There are two types of smart pointers: `std::unique_ptr<T>` and `std::shared_ptr<T>`.

`std::unique_ptr<T>` indicates exclusive ownership of the instance pointed at. Therefore, unique pointers cannot be copied, as this would result in an unclear ownership situation. When the unique pointer goes out of scope, it will call `delete` on the contained instance. This prevents the possibility of memory leaks. It is perfectly safe to retrieve raw pointers `T*` from the unique pointer, as long as those are not explicitly deleted. In this case, care has to be taken to not dereference the raw pointer after the unique pointer has deallocated the memory, exactly as if working purely with raw pointers. It is also possible to release the unique pointer, converting it into a raw pointer, and preventing the automatic deletion.

`std::shared_ptr<T>` can be used to model shared ownership, by implementing the classic reference counting paradigm. In essence, the class prefixes the memory block used to store `T` with an integer. This integer is incremented, whenever the shared pointer is copied, and decremented whenever any instance of the shared pointer goes out of scope. Only when the decremented instance count reaches zero, a `delete` is issued, since this situation means that no shared pointers pointing at the given object remain. As with the unique pointer, a shared pointer can also be queried for a raw pointer, which can then be used regularly, with the same caveats as for the other type. Shared pointers cannot release their contained pointer, since that would mean having to retrieve all copies of the shared pointer, and communicating that the pointer was released.

While `std::unique_ptr<T>` is thread-safe automatically, since it cannot be copied, shared pointers need to explicitly synchronize access to the reference count. This is necessary to ensure that multiple threads can independently copy and delete shared pointer instances,

while making sure that the final instance will issue the deallocation. This is achieved by using an atomic integer for the reference count, which has the intrinsic property that the increment and decrement operations are thread-safe. On the other hand, atomicity comes at a performance cost, since special CPU mechanisms need to be employed to ensure the consistency. As a consequence, copying `std::shared_ptr<T>` is relatively expensive, and should be avoided as much as possible.

The ACTS version of `Surface` has been updated to use `std::shared_ptr` during the EDM-related work for this thesis. This is done in multiple steps:

1. All surfaces are allocated as shared pointers. To ensure this, the constructor is made private, effectively disallowing unmanaged heap allocation, and even stack allocation. Allocation is only possible via

   ```
   template <class T, typename... Args>
   static Surface::std::shared_ptr<T> makeShared(Args&&... args);
   ```

2. Wherever no explicit ownership is required, i.e. if surfaces are not stored for later use, raw pointers or references to the surfaces are used. This avoids the expensive copy operations, while still allowing access to the surface. In rare cases, references to the shared pointer itself are used.

3. Since all surface allocations are managed, a special function can be used to retrieve the corresponding shared pointer for a given raw pointer:

   ```
   std::shared_ptr<Surface> Surface::getSharedPtr();
   ```

   This allows taking partial ownership of a surface, even if a shared pointer had not explicitly been provided.

The additions to the `Surface` class have been added to the ACTS code, all usage of unmanaged allocation was removed in favor of the `Surface::makeShared` factory function. Code that explicitly had to check the surface ownership status, and perform clones was replaced by simple shared pointer copy operations. The ownership is therefore handled automatically, and only minimal explicit care has to be taken. Due to the aggressive use of raw pointers in combination of the sparing use of the shared pointers themselves, only few of the expensive copy operations need to be performed. Therefore, no performance degradations were observed, relative to the manual ownership model.

7.7. Concurrent handling of time dependent parameters such as alignment

7.7.1. Time dependent misalignment

As was previously explained in section 4.10, the exact positions and orientations of the sensors of the tracking systems do not stay constant over time. Due to effects like changing

temperatures or properties of the cooling systems or the magnets, sensor assemblies and even individual sensors move. In terms of track reconstruction, this results in the assumed location of the sensor being incorrect, which has an impact on the comparison between the predicted trajectory position, and the measurement position. As a consequence, the precision of the extracted track properties can be negatively affected, if not corrected for. Therefore, ATLAS uses a strategy to measure and correct for these misalignments. Special attention is directed to the short-timescale deformation of the IBL depending on its temperature. Alignment corrections are calculated for time-intervals as short as 60 s.

7.7.2. Conditions handling in ATLAS

Time dependent information such as the alignment corrections discussed in the previous section, are referred to generically as *conditions data* in ATLAS. Apart from explicitly calculated corrections such as the alignment, it also includes measured data such as temperatures, exact high voltages and similar data. They model the condition of the detector over time, hence the name. Conditions data is stored in dedicated time-series databases, the granularity of the stored information depends on the frequency with which it is being written. Some data is written during data taking itself, while other data is calculated after the fact and written retroactively.

Conversely, conditions data also needs to be applied during reconstruction. In the ATLAS software, access to the conditions databases is managed by dedicated code that knows how to connect to the data sources and is able to do various forms of caching to improve performance. What remains is to implement reading the conditions data through the access layer wherever it is needed, and use it.

In a single-threaded execution model, this setup is relatively simple. Processing occurs event by event. A special service exists, that is responsible for handling *incidents* that occur. One example for such an incident is when the processing reaches the boundary between Intervals Of Validity (IOVs) of the conditions data. Other components can register themselves with the incident service, in order to be notified when an incident is encountered. The component can perform arbitrary operations upon being notified. As illustrated in figure 7.52, when an IOV boundary incident is dispatched, this opportunity can be used to update the conditions data. Since the incident is triggered when the new conditions data is already available, these components can simply read the information from the central conditions service.

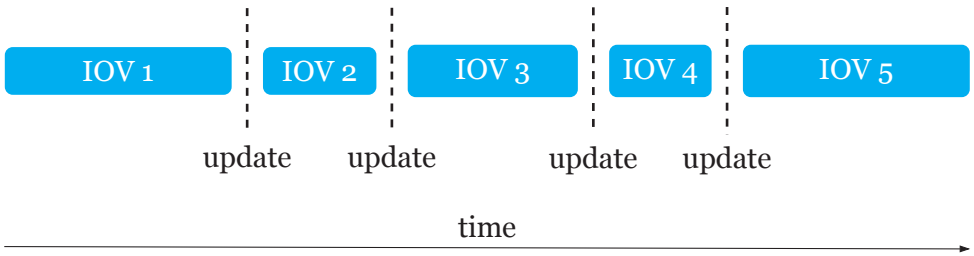


Figure 7.52.: Illustration of a sequence of IOVs, where an update occurs at the boundary between two neighboring IOVs.

In the context of the alignment corrections this is exactly what happens. As previously mentioned in section 7.2, the `GeoModel` hierarchy contains a dedicated type of transform object that can be configured with a correction transform, called *delta*, to be applied in addition to the nominal transformation. During the initial construction, these deltas are set to zero, meaning a nominal geometry is constructed. Whenever an IOV change occurs that concerns the alignment, the detector manager, which owns the hierarchy, traverses every node, and updates the alignable transforms. It also invalidates any caches that do not apply anymore, since the alignment changed. When the update pass completes, normal processing is resumed. All new caches being built use the updated alignment deltas, and are therefore correct.

The different levels of alignment can be processed nearly independently of one another. As the update pass is able to only invalidate transform caches that actually changed, the multi-leveled alignment deltas do not incur a significant overhead. The high frequency Level 3 changes for the IBL, for instance, only affect individual sensors, which means only their transform caches become out of date.

As previously mentioned, the ATLAS software is moving to a new multi-threaded execution model (see section 3.3.2). A consequence of this is that multiple events can be processed concurrently. In turn, this means that the clear boundaries shown in figure 7.52 vanish. Instead, it is not sufficient to have one complete valid instance of the geometry: every IOV that has active events being processed needs access to a geometry instance.

From an infrastructure perspective, a number of changes are required. Rather than using the central conditions service for reading, a more flexible approach is used. Algorithms and tools can register conditions data dependencies. On the other hand, special conditions algorithms can be configured, that explicitly advertize which type of conditions data they are able to provide. The multi-threaded scheduler reconciles the requests for conditions data with the advertized data, and schedules the conditions algorithms, such that they run before any components that depend on their output. A dedicated conditions store is used to support the data transfer. Data in the conditions store is required to be immutable, to prevent race conditions.

Concurrent conditions access then works by using a *conditions handle* alongside a *conditions handle key*. The latter is used to declare the previously mentioned conditions data dependencies of a component. The handle is constructed using the handle key, while explicitly providing an event context, which encodes the event's location in the sequence of IOVs.

7.7.3. Concurrent handling of alignment corrections

In the context of the multi-threaded execution model, the concurrent conditions handling also affects the alignment corrections. Since the conditions data is stored in the conditions store, and is immutable, using transform caches inside the geometry hierarchy and invalidating them when applying new corrections becomes impractical. This problem can be solved in two different ways:

1. Rebuild or copy the full geometry. Either use the correct alignment data during the

rebuilding process, or apply the updates during the copy process.

2. Reuse the same instance of the geometry. Externally provide the variable alignment deltas for the current IOV.

In the ATLAS tracking library, the former approach is chosen, as requires fewer intrusive changes to the codebase. Before the migration, the ATLAS tracking geometry service constructed the tracking geometry from the collection of readout detector elements. This collection, in turn, is based off of the detector hierarchy described in `GeoModel`. As a first step, `GeoModel` was updated to optionally accept an auxiliary object `GeoAlignmentStore` wherever a transform is accessed. Rather than internally keeping caches, they are stored in the alignment store instead. By explicitly passing the alignment deltas, the client can provide them based on the event context, in a thread-safe way.

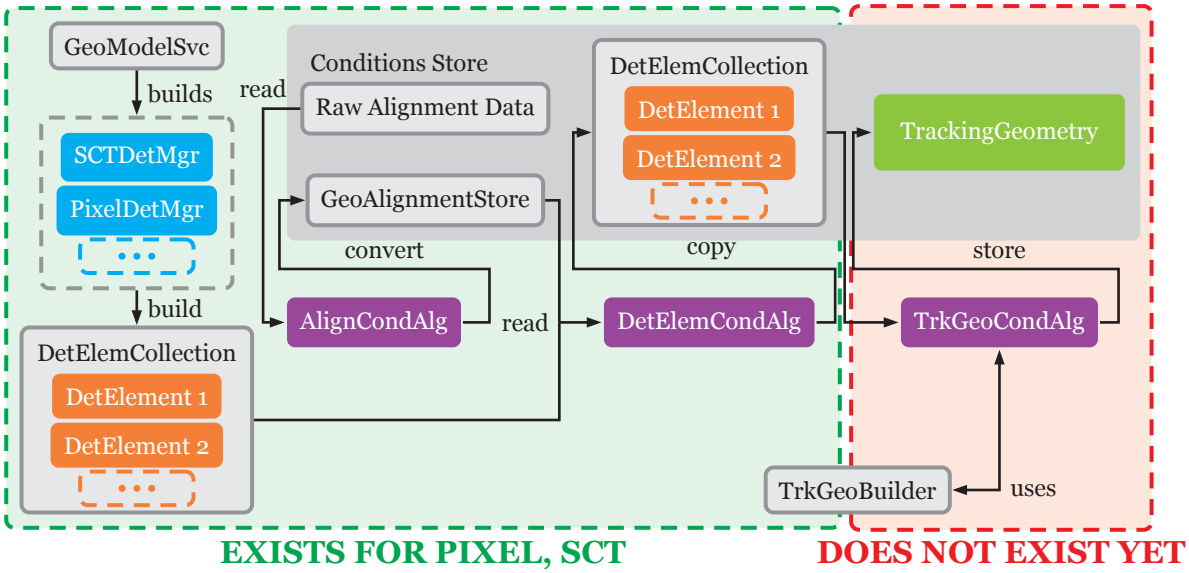


Figure 7.53.: Illustration of the workflow that provides geometry data including the correct alignments. Shown are a number of algorithms that produce the `GeoModel` geometry, the readout geometry as well as alignment stores containing the actual delta matrices. On the right side, the planned strategy for the implementation a multi-threaded tracking geometry with alignment that lives in the conditions store is outlined.

Figure 7.53 shows an overview of the conversion steps from raw alignment data to usable geometry information. Only some of the interrelationships are shown for clarity. Initially, as discussed previously, the `GeoModel` service constructs the geometry hierarchy, and creates one detector manager per subsystem. This is shown in the top left part of the figure. Each detector manager builds a collection of readout detector elements, which it augments with additional information, visible in the lower left part. In the single-threaded execution model, this collection would iteratively be updated as alignment deltas change. In the concurrent model, however, the collection is considered *nominal*, and does not change. An alignment conditions algorithm (*AlignCondAlg*) is scheduled and reads the raw alignment data from

the conditions store. It uses this information to construct a `GeoAlignmentStore`, which contains all the alignment deltas in a format that `GeoModel` understands. This procedure completes the left half of figure 7.53. Another conditions algorithm (*DetElemCondAlg*) reads both the alignment store as well as the nominal readout detector element collection to produce an updated copy, which is also stored in the conditions store, as seen in the figure.

As a next step, the tracking geometry needs to be made aware of the new alignment infrastructure. This is not currently implemented in the ATLAS tracking library. To limit the amount of changes to the existing code, the plan is to replicate the previous steps, by adding yet another conditions algorithm (*TrkGeoCondAlg*) which reads the readout detector element collection and reruns the tracking geometry construction process. After construction, the new tracking geometry is again written to the conditions store. Figure 7.53 indicates this part of the strategy to not exist yet. The exception is the *TrkGeoBuilder* tool, which partially exists, but needs to be updated such that it can be called from *TrkGeoCondAlg*.

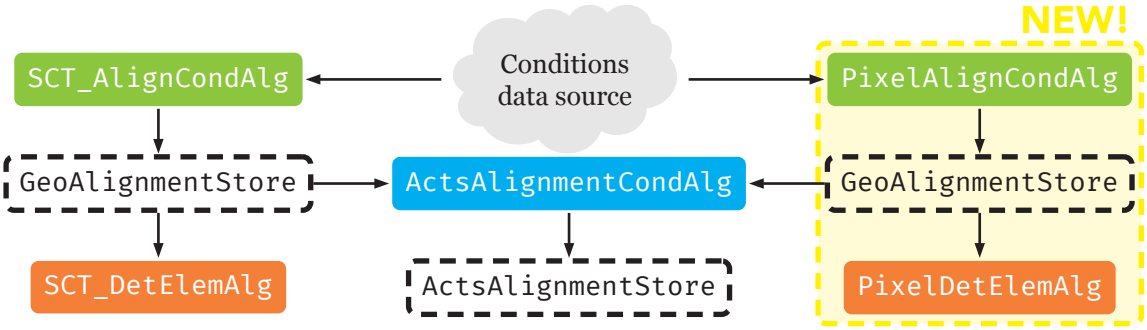


Figure 7.54.: Overview of how the different components of the concurrent alignment infrastructure interact with each other, and with ACTS. Also indicated is new functionality added to the ATLAS software to support the setup.

As the conversion process depends on the specifics of the subsystem, it has to be implemented using dedicated algorithms for each of them. The overall conversion chain is displayed again at a high-level in figure 7.54. The SCT variants of the conditions algorithms were added prior to the work for this thesis. In the context of this thesis, equivalent algorithms for the Pixel detector were added. These new algorithms are based on the existing single-threaded mathematical implementation of the alignment, and reuses some of the alignment calculation infrastructure. As shown on the right side of figure 7.54, they mirror their SCT counterparts. The main difference is that they explicitly handle the IBL bowing, whose corrections are given in a different format and need to be converted carefully. A comparison between the aligned transforms of the updated copy of the readout detector element collection and the ones obtained in the single threaded mode was conducted. The output of the single-threaded execution uses the preceding working model of the alignment updates, and is considered as a baseline. Identical results were achieved using the new implementation. Since ACTS, by design, has no real concept of different subsystems, the separate alignment stores for the SCT and Pixel subsystems are merged into a combined `ActsAlignmentStore` by another conditions algorithm. This is indicated in the central part of figure 7.54.

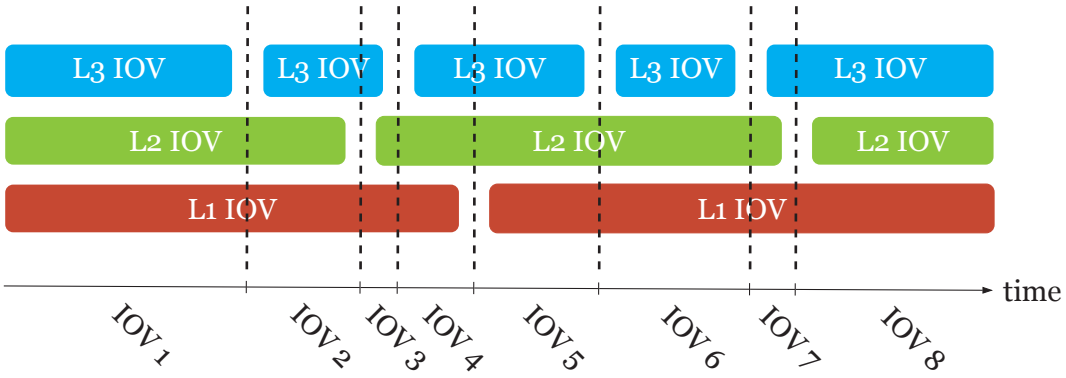


Figure 7.55.: Illustration of how IOVs with varying lengths can overlap. Shown are the alignment levels 1 through 3, where Level 1 changes most frequently, while Level 3 changes less often. In the lower part, the effective IOVs, resulting from combining the three levels are shown. The frequency is essentially dominated by the conditions data with the shortest IOVs.

One particular issue with this approach is due to the fact that the immutable data in the conditions store needs to be complete. This is addressed by fully populating the readout detector element collections in the conditions algorithms discussed before. As a result, rather than being able to only update the parts of the geometry that are affected by the changed alignment level, all alignment deltas need to be applied. Consequently, the IOVs have to be merged, to provide one set of effective IOVs for the entire alignment. Figure 7.55 shows the effect of this circumstance. The Level 3 IOVs change with the highest frequency. Shown are the three levels of alignment in different colors. It also shows the effective IOVs that are the result of the combination of the three alignment levels. Their overall frequency of change is dominated by the short-lived Level 3 alignment. As a consequence, large parts of the geometry construction sequence need to be rerun at the Level 3 frequency, which can be as low as 60 s.

The second of the two solutions mentioned at the beginning of this chapter, which alleviates these problems, was devised for this thesis. The basic idea is to directly query the alignment store whenever the transform for a sensor is requested. To enable this, the sensor needs to obtain a reference to the alignment store that is valid for the IOV the event currently being processed belongs to. With this approach, the need to copy the tracking geometry vanishes, clients can always refer to a single instance of the geometry that will present correctly based on the event context.

The location for such experiment specific code within ACTS is the detector element class (see listing 7.1 on page 118), which `Acts::Surface` instances can be linked with. Its method `transform` delegates to the detector element object, if one is set, and returns the sensor transform. This method can be used to retrieve the alignment dependent transform and return it to the caller transparently. Alignment deltas therefore need to be retrieved from an alignment store inside the `transform` method. Two approaches to obtain an alignment store were considered: one where the detector elements does a lookup on its own, and one where the alignment store is explicitly provided.

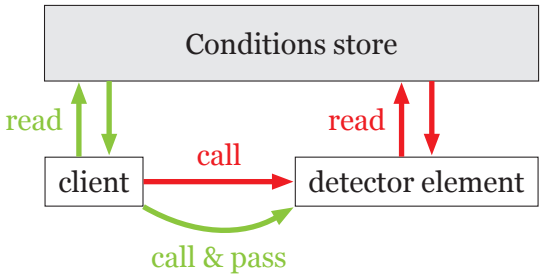


Figure 7.56.: Illustration of the two methods to make alignment information accessible from the detector element object. The method where the detector element communicates with the conditions store directly is shown in red, while the method where information is passed explicitly is shown in green.

The first approach is complicated by the fact that conditions handle keys only work correctly in two of the three types of software components used in Athena: algorithms and tools. As opposed to services, these types of components participate in the scheduling. To enable the detector element to be able to retrieve the alignment store, the tracking geometry service was used as a mediator to remember the currently valid alignment store for each thread. Detector elements can then query the service for the alignment store, and retrieve the correct transform. Figure 7.56 shows this workflow in red, where the detector element essentially directly communicates with the conditions store.

The second approach avoids the explicit lookup via the tracking geometry service inside the detector element in favor of providing the alignment store explicitly through function arguments. To support this in an experiment independent way in ACTS, all call chains that lead to transform queries had to be modified to accept an additional argument that can be used for this purpose. This *geometry context* is implemented using the `std::any` type from the C++ STL, so that arbitrary information can be passed. ACTS, by design, never inspects the contents of this context object, and only passes it along. Using this infrastructure, the ATLAS aware detector element in Athena was modified to unpack the combined alignment store from provided context object. The alignment store can then be queried for a transform directly, without the need to retrieve the event context first, or use thread-local storage. This workflow is indicated in green in figure 7.56, where the client communicates with the conditions store, and passes the information along to the detector element. Due to the conceptual complexity, and the potential performance overhead of having to use thread-local storage, and frequent event context lookups in the transform queries to the detector elements, the latter approach was favored over the former.

While it is difficult to determine whether clients see exactly the correct alignment, a sanity check was conducted. To this end, a specially prepared dataset, which contains a collection of events across a large number of IOVs was prepared. Running over this input dataset, for every event the tracking geometry was written to disk in the Wavefront OBJ format previously discussed in section 7.2.3. The processing was configured to run on all available CPU cores concurrently, and care was taken to not cause race conditions while writing to disk. Events

from the same IOV were found to produce identical OBJ outputs. In order to be able to visualize any differences in alignment, the OBJ output was modified after being written out. In each file, all surfaces are compared to their equivalent from the first processed file. This first file is therefore treated as a sort of nominal reference. The obtained difference was then scaled up, and a shift was applied to all vertices along the normal for each surface. The reason for applying this exaggeration at this stage is that the nominal reference from the first file is needed to produce visible variations. Simply scaling up the alignment deltas directly in the software exaggerates the alignment effect itself, but will not yield sufficient deviations between the IOVs that are shown.

The results of this visualization are shown in figures 7.57 to 7.59 for the Pixel and SCT detectors, respectively. Figure 7.57 shows the endcaps of the pixel detector. Sensors aligned according to different IOVs are overlaid and colored individually. The IOVs shown here are from a sequence of data taking runs. Different IOVs appear to be coherently separated. Figure 7.58 shows the IBL, the innermost barrel layer of the Pixel detector. As discussed previously, the IBL stave bow: at the start of a run, they are approximately straight, as the run goes on, they start to bow along the radial direction. In the figure, the colors indicate a sequence of LBs from a single run. In green, toward the beginning of the run, the bowing is minimal. It is clearly visible in red, after the bowing has saturated.

Finally, figure 7.59 shows an equivalent picture for the endcap disks of the SCT detector. Again, the sequence of IOVs result in sensors that are separated from each other. The IOVs are again from different runs in this image. The overlap between the yellow and red IOVs is due to the fact that the exaggerated deltas are effectively arbitrary, and no precautions are in place to prevent overlaps. Without the exaggeration, no overlaps occur.

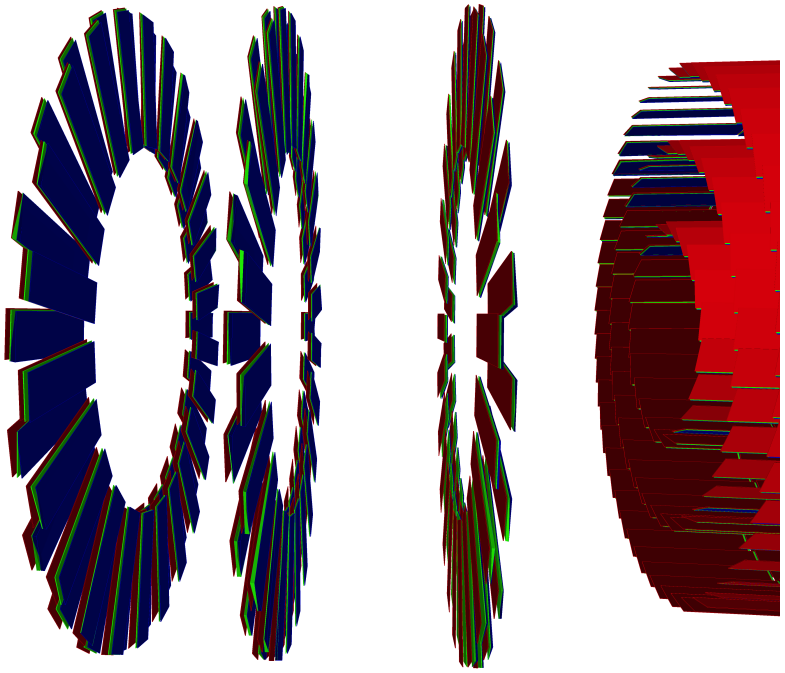


Figure 7.57.: Exaggerated display of the misalignment of the Pixel detector. Shown are the endcaps and parts of the barrel. The different colors are from different IOVs across multiple data taking runs.

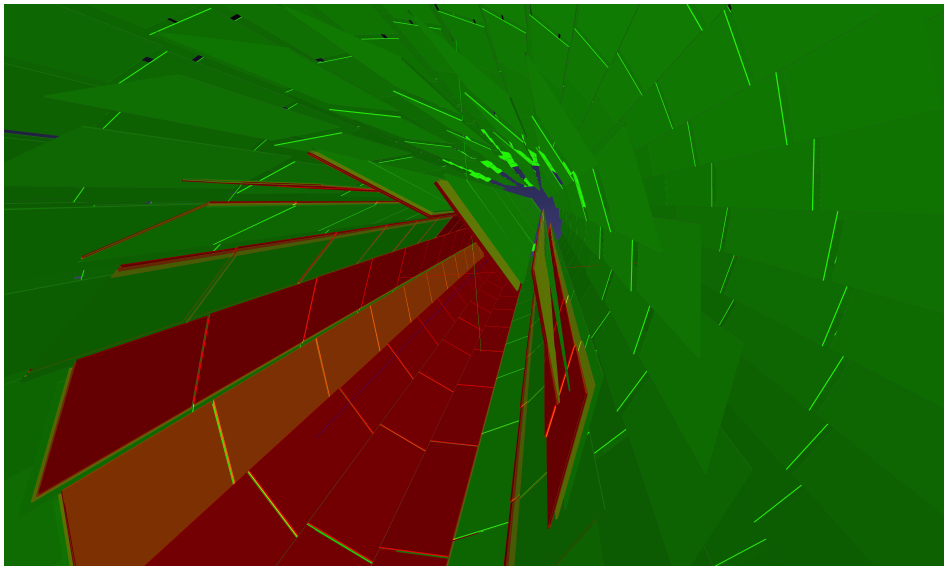


Figure 7.58.: Exaggerated display of the misalignment of the Pixel detector. Shown is the misalignment of the IBL. The different colors indicate IOVs from a single run, across a number of LBs. In the very center, the bowing of the IBL staves is seen. It is minimal in the first LB of the run in green, and increases toward the last LB shown in red.

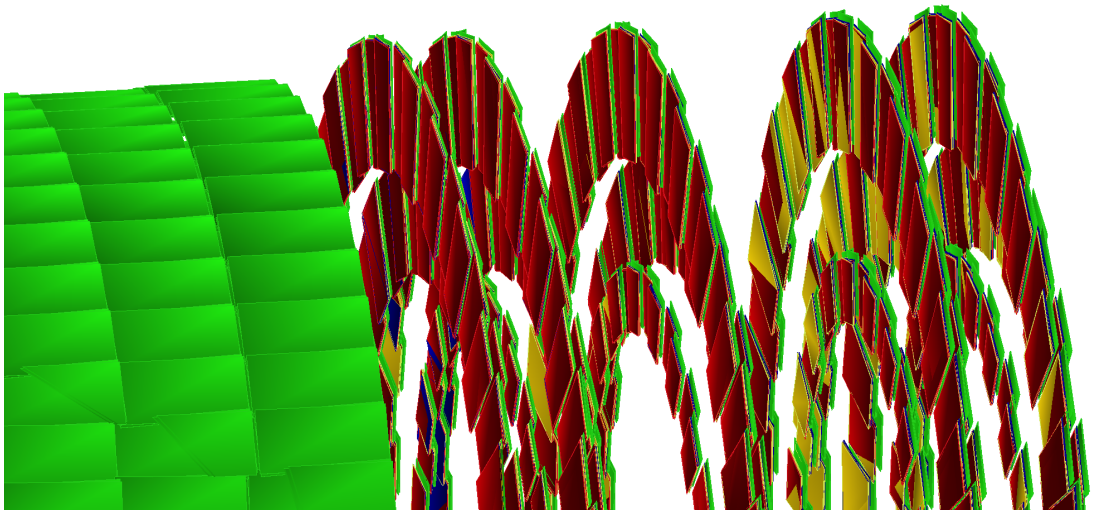


Figure 7.59.: Exaggerated display of the misaligned sensors in the endcaps and part of the barrel of the SCT detector. The different colors indicate different IOVs across a number of runs.

7.7.4. Component structure

In order to facilitate usage of the ACTS geometry and extrapolation from within the ATLAS software, the structure of components was adapted to model the alignment information propagation workflow discussed in the previous section. As mentioned in section 7.2.3, a special service constructs the ACTS tracking geometry. A dedicated `ActsTrackingGeometryTool` was added, which provides a method to access the tracking geometry owned by the service. Having access to the geometry through a tool has the advantage that the it can declare an input dependency for the alignment conditions data. This means that the required conditions algorithms automatically get scheduled when a client is set up to access the tracking geometry through the tool. Also featured is a method to retrieve an alignment store that is retrieved from the conditions store, using a provided event context. This alignment store is directly returned in the form of an ACTS geometry context, ready to use with other components.

Another provided tool is the `ActsExtrapolationTool`. It sets up and exposes the ACTS propagation infrastructure. By using the tracking geometry tool from above, it is able to automatically handle the alignment. Aside from that, a component that connects the ACTS propagation with the ATLAS magnetic field data source was implemented.

The provided components allow easy and complete access to the ACTS tracking geometry description, automatically and correctly takes care of alignment corrections. However, further developments to make more features of ACTS available in a convenient way from within Athena is still required.

7.8. Summary and outlook on work to be done

In this chapter, various aspects of ACTS development that were undertaken in the context of this thesis were outlined and discussed. A point of focus was the integration of ACTS components into the ATLAS software environment.

Section 7.1 introduced the infrastructure, that was added in order to enable providing the ACTS library to code in ATLAS. Various packages were added that interconnect between the library and the framework.

Section 7.2 contained a description of how the current geometry of the ATLAS ID was converted into a format that ACTS understands and can use. Details on how the information flows and which components are responsible for which part of the conversion were given. Finally, an extrapolation test, were pseudo-particles are traced through the geometry was shown to validate that the geometry and navigation works correctly. Work remains in order to optimize the modelling of the TRT detector. A geometry description of the ATLAS MS has yet to be developed.

Section 7.3 presented the geometry conversion for the ITk Phase-2 upgrade of ATLAS. A feasibility study into whether the overall structure of the geometry can be described in ACTS was shown. A difference between the current and the future geometries was discussed in detail: the endcap sensors of the ITk strip system. In the form of a new implementation, the sensor was shown to be described well using a polar coordinate system. Details and characteristics of the new implementation were given and compared with the existing shape setup. A demonstration that this new shape can be correctly handled by the ATLAS KF was given, but a full integration has yet to be carried out.

Section 7.4 discussed in detail a completely new navigation model catered toward irregular geometries, such as the one found in the ATLAS calorimeters. It uses intersections of rays and frustums with hierarchies of bounding boxes, to enable fast retrieval of navigation candidates from a very large collection of calorimeter cells. To enable testing, a description of the ATLAS calorimeter readout geometry was implemented for ACTS. The navigation model was demonstrated to work as expected, and its performance was characterized. All components for the navigation model are assembled, but a substantial rewrite of the ACTS navigation driver will be required to make full use of thew new approach.

Section 7.5 showed several aspects of how metaprogramming can be used to increase flexibility and performance of the EDM. A new column-based memory storage backend for filtering applications was introduced, and the benefits and tradeoffs associated with it discussed in detail.

In Section 7.6 a new approach for dealing with memory allocation of shared surface objects was presented. It uses modern C++ facilities to remove manually implemented management code, that was susceptible to problems.

Finally, section 7.7 concerned itself with the first implementation of alignment corrections in the context of the multi-threaded execution model of the ATLAS software. It introduced the complexities of alignment in this environment, and showed how they can be solved in

a experiment independent generic way on the ACTS side, and a direct integration with the alignment infrastructure on the ATLAS side. The concurrent alignment setup was found to correctly propagate the required alignment information without synchronization. It is currently applicable to the alignment of the Pixel and SCT detectors, but has yet to be extended to also include the TRT

In conclusion, the chapter showed significant advancements toward integration of ACTS in ATLAS. During the work for this thesis, the basic integration model was defined and implemented. Various interactions between the two software domains were reworked and adjusted, in order to fit the requirements. This work does and will serve as the baseline, upon which the integration effort continues, and established guidelines and best practices for how it can be accomplished. As the range of tracking applications in ATLAS is very large, the integration effort is still ongoing.

Part III.

Search for long-lived particles presenting as disappearing tracks

8. Disappearing tracks in the ATLAS detector

8.1. The disappearing track signature

This chapter focusses on a particular type of signature that can be observed in the ATLAS detector as a result of long-lived particles. Long-lived, in this regard, refers to particles whose lifetime is long enough for them to leave the immediate interaction region before decaying. In contrast to shorter-lived particles that are part of the prompt proton-proton interaction, and can therefore only be investigated through their decay products, these particles interact with the sensors themselves, opening them up for detection.

Depending on the processes under study, these long-lived particles can result in various types of signatures, depending on their concrete lifetime. If the lifetimes are comparatively large, but not long enough for the particle to be considered stable, their decay can result in vertices that are far away from the interaction region. In some SUSY scenarios, such as the one that was discussed in section 2.3.1 and is the center of attention in chapter 9, the expected range of lifetimes is long enough for the long-lived particle itself to traverse the region of the ATLAS Pixel detector, but short enough for it to decay before reaching the SCT.

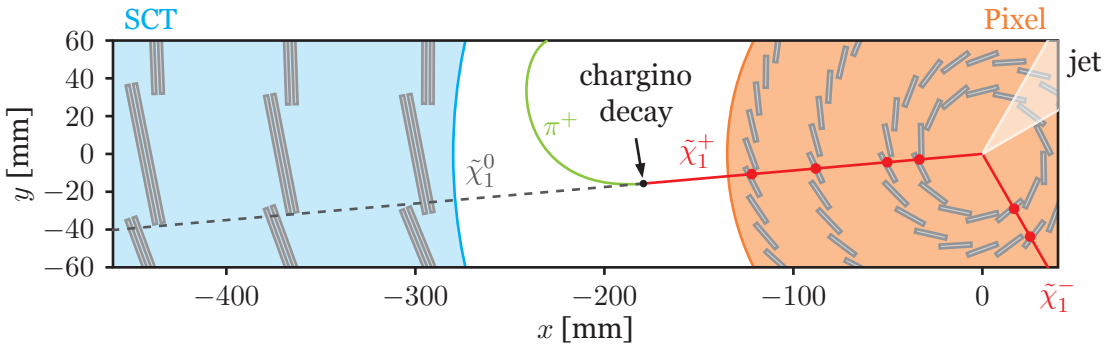


Figure 8.1.: Sketch of a chargino decay in the transverse plane of the ATLAS detector. A pair of oppositely charged $\tilde{\chi}_1^\pm$ is produced in the center, alongside a jet. The positive chargino moves through the Pixel detector volume before decaying in the space between the Pixel and the SCT.

Figure 8.1 shows an illustration of this type of signature. At the interaction region on the right, a pair of $\tilde{\chi}_1^+\tilde{\chi}_1^-$ is produced alongside a jet. The charginos recoil off of the jet, thereby obtaining transverse momentum. With a large enough lifetime of $\mathcal{O}(\text{ns})$, the chargino reaches

the gap between the Pixel and SCT detectors before decaying, for example into a neutralino and a charged pion $\pi^{\pm}$, at about -180 mm. The neutralino does not interact and is stable, if R-parity is conserved and it is the LSP. Therefore, the neutralino escapes the detector undetected in this scenario. In most cases, the pion is not detected either, as its momentum is generally very low. However, the chargino will itself have left a signature in the innermost part of the detector: a short particle track that *disappears*! This short type of track is referred to as a *tracklet* in the rest of this thesis.

These tracklets are generally not reconstructed as part of the standard algorithms and configurations, since they are mostly of interest for very specific purposes. As a consequence, the entire decay chain of the charged short track, the charged pion, as well as the neutralino are invisible to the standard reconstruction. As discussed in section 3.5.4, dedicated algorithms exist to detect imbalances in the transverse energy observed in an event. In the event topology described here, and shown in figure 8.1, a recoil jet causes the undetected portion of the process to have non-negligible transverse momentum, therefore contributing to the missing transverse energy.

8.2. Four- and three-layer tracklet reconstruction

The disappearing track signature consists of a short tracklet in the innermost part of the detector, that vanishes when the underlying charged particle decays into exclusively invisible daughter particles. In order to mount a search for these types of events, dedicated track reconstruction techniques have to be deployed to try to reconstruct these short tracklets.

Standard tracks

Standard track reconstruction as described in chapter 4 for ATLAS in particular, is optimized toward conventional tracks caused by stable particles which traverse all of the ID subsystems, the Pixel, SCT and TRT detector. In a first stage, triplets of space points are created from one or more measurements and filtered based on general compatibility with the interaction region. Both triplets of Pixel and triplets of SCT hits are considered. This is done by calculating approximate parameters based on the assumption of a helical trajectory in a constant magnetic field. At this stage, requirements such as on the transverse momentum $p_{\mathrm{T}} > 500$ MeV, as well as the longitudinal and transverse impact parameters $|d_0| < 10$ mm, $|z_0| < 250$ mm are enforced. These impact parameters are calculated with respect to the beam spot, rather than the primary vertex (PV), as the latter is not yet available at this stage. This results in significantly worse impact parameter resolution [158].

The seeds are then processed in a track finding stage, during which a CKF collects any additional measurements that are compatible with the trajectory. Missing measurements on sensors where they are expected are recorded as *holes*. If more than two of these holes are found, the track is discarded. A minimum of seven silicon hits are required for track candidates to proceed. Subsequently, an ambiguity resolution algorithm scores and ranks

track candidates based on their length, number of holes, measurements shared with other candidates and additional detailed hit information. Tracks which pass this stage are then extrapolated into the TRT to collect any additional measurements the underlying particle might have caused there.

Pixel tracklet reconstruction

This section gives an outline of the specialized tracklet reconstruction procedure detailed in [217]. It is executed in a second pass after the standard tracking. Measurements which are assigned to any of the tracks that pass the standard selection criteria are removed from the pool of eligible measurements for this pass. This means that any standard tracks have priority to be assigned available hits over tracklets. Again, triplets are formed from spacepoints, however, in this case only Pixel hits are considered. Any trajectories that would have enough SCT hits for a triplet will already efficiently be picked up on by the first pass. The Pixel triplets are subjected to the same requirement as standard seeds, with additional requirements of $p_{\mathrm{T}} > 5\,\mathrm{GeV}$ and $|\eta| < 2.2$, to suppress seeds caused by random combinations and restrict track finding to the central parts of the ID.

During the subsequent track finding phase, a more stringent requirement on the χ^2 between the measurement and predicted sensor intersection is set. The track finding is also not allowed to branch out in case of multiple measurements, but rather selects the one which fits best. Trajectories with missing hits are rejected. An ambiguity resolution stage that is equivalent to the standard reconstruction follows, with the minimum requirement of seven silicon hits reduced to three.

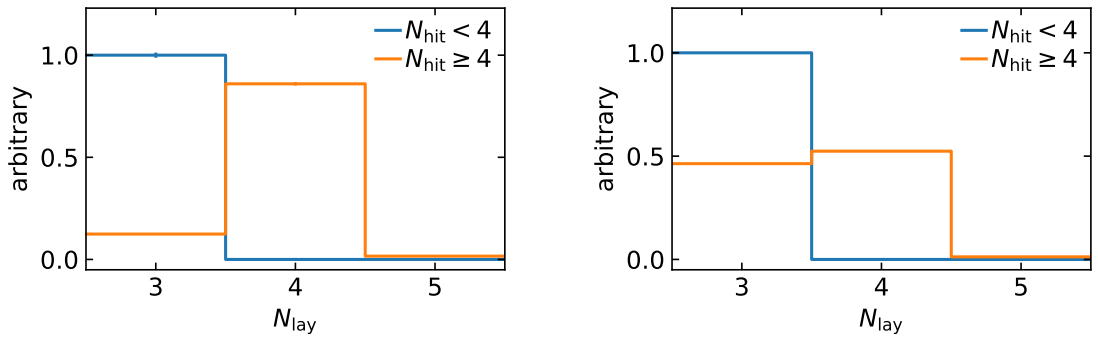
Previous ATLAS publications used variations of the reconstruction procedure described here. A search using the full LHC Run 1 dataset of an integrated luminosity of $20.3\,\mathrm{fb}^{-1}$ at $\sqrt{s} = 8\,\mathrm{TeV}$ [218] used tracklets consisting of at least three Pixel hits and at least two SCT hits and predated the installation of the IBL. A following search using $36.1\,\mathrm{fb}^{-1}$ of integrated luminosity taken at $\sqrt{s} = 13\,\mathrm{TeV}$ during Run 2 [219] required at least four Pixel hits, and was therefore able to achieve sensitivity to much lower lifetimes than the Run 1 analysis, resulting in shorter tracklet lengths. This analysis also introduced a veto of tracklets having an associated SCT hit, which is applied after the reconstruction phase at the analysis level.

Modifications were made to the reconstruction procedure to relax this requirement to three Pixel hits. This reduction results in an increase of random hit triplets passing the selection criteria, which manifests itself in larger background yields. A large portion of Pixel-only tracklets is assigned a random SCT hit during reconstruction. This is simply due to the high likelihood of there being an available SCT hit, possibly originating from pile-up interactions, that loosely fits the tracklet. Since the analysis-level SCT hit veto will remove these tracklets, this aspect presents a significant problem for the reconstruction efficiency of a chargino decay. As a consequence, good tracklet candidates are removed due to the random additional hit, that would otherwise be usable. To combat this, a hit filter was introduced to attempt to recover these candidates. During the track finding, SCT hits are only attached to the trajectory, if the second-to-last measurement is not on a Pixel layer. Additionally, in case multiple SCT hits

are attached, they are required to be on the same layer and on pairs of stereo-strip modules (see section 3.2.2).

Reconstruction performance

In order to understand the impact and viability of the reconstruction techniques outlined above, it is convenient to look at the reconstructed tracklets from a signal process producing disappearing tracks. The following uses a set of benchmark MC samples (see table B.1) with $m(\tilde{\chi}_1^{\pm}) \in \{160, 200, 240\}$ GeV, all generated at a chargino lifetime of $\tau(\tilde{\chi}_1^{\pm}) = 0.2$ ns.



(a) truth-matched

(b) not truth-matched

Figure 8.2.: Normalized distributions of the number of contributing layers toward a tracklet N_{lay}, for tracklets with $N_{\text{hit}} = 3$ and $N_{\text{hit}} \geq 4$. The left plot shows tracklets which are matched to a true chargino, while the right shows tracklets for which this is not the case.

In figure 8.2, the distribution of the number of Pixel layers contributing toward the reconstruction of a given tracklet is shown. It is further divided into tracklets which are reconstructed from $N_{\text{hit}} = 3$ and $N_{\text{hit}} \geq 4$, and tracklets matched to charginos are shown separately from ones that are not. N_{lay} and N_{hit} do not need to be identical, as there is a possibility for a tracklet to include two measurements on a single layer. This is possible, since the sensors on each layer slightly overlap at their edges, as shown in figure 8.1.

For $N_{\text{hit}} = 3$, the number of contributing layers cannot be larger than three. Having three measurements from only two layers is not desirable due to the very short track segment length covered by the measurements. For tracklets with four or more measurements, the distribution shows that a non-negligible fraction of tracklets are reconstructed from only three layers. This is more so the case for tracklets which are not matched to true charginos (figure 8.2b) than it is for the ones that are (figure 8.2a).

In the rest of the description of the tracklet properties, N_{pix} will refer to the number of Pixel layers contributing hits toward a tracklet. As can be seen above, such a tracklet can include multiple hits from a single layer. Since the properties of the tracklet are dominated by the number of layers, rather than the number of hits, the former quantity is used to discriminate between tracklet types.

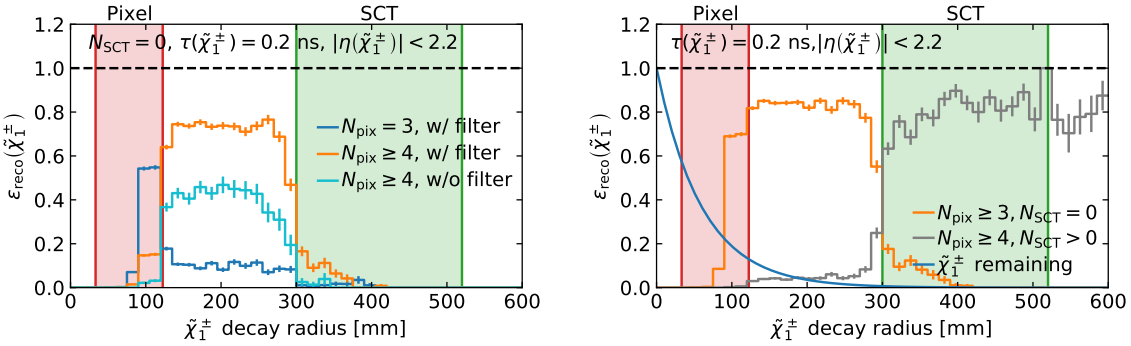
Reconstruction efficiency

Another critical quantity is the reconstruction efficiency $\epsilon_{\text{reco}}(\tilde{\chi}_1^{\pm})$. It is a measure of how likely a signal chargino particle is reconstructed successfully as a tracklet and can be defined as

$$\epsilon_{\text{reco}}(\tilde{\chi}_1^{\pm}) = \frac{\text{number of } \tilde{\chi}_1^{\pm} \text{ with a matched tracklet}}{\text{number of generated } \tilde{\chi}_1^{\pm}}. \tag{8.1}$$

The higher this fraction is better the tracklet reconstruction performs at finding the chargino signature in the event. This ultimately manifests itself in a better sensitivity for a signal process producing these charginos and signatures, although other aspects, such as the rate of background events, also play an important role.

Figure 8.3a shows the reconstruction efficiency as a function of the true radius of the chargino decay vertex in the transverse plane. It uses the same simulated samples from before, the histograms shown are a combination of samples with $m(\tilde{\chi}_1^{\pm}) \in 160, 240$ GeV. Also shown are the volumes covered by the Pixel and SCT detectors from 33.25 mm-122.5 mm and 299 mm-514 mm, respectively. To suppress the effect of the seed-level tracklet selection criterion of $|\eta| < 2.2$ for this figure, the set of eligible charginos for the ratio is restricted to $|\eta| < 2.2$.



(a) (b)

Figure 8.3.: Distributions of the chargino reconstruction efficiency for various selections of tracklets as a function of chargino decay vertex. Shown in (a) are three-layer and four-layer tracklets, the latter are drawn both including and excluding the SCT hit filter. (b) shows all tracklets with $N_{\text{SCT}} = 0$ in comparison with those with $N_{\text{SCT}} > 0$, as well as the exponential decay of a chargino with a rest frame lifetime $\tau(\tilde{\chi}_1^{\pm}) = 0.2$ ns and neglecting time dilation.

The histogram in light blue represents the previous reconstruction configuration, requiring at least four Pixel hits, and excluding the SCT hit filter. It is nearly zero up to the edge of the Pixel detector, before rising to about 40 %. The SCT veto causes a sudden drop at the beginning of the SCT volume, as decay vertices at this radius are more likely to feature SCT hits. On the other hand, the orange histogram uses the new reconstruction version including the filter. It can be observed to almost double the reconstruction efficiency in the gap region

between the two silicon detectors, while also featuring a non-zero efficiency in the inner part of the SCT volume. The histogram indicates no significant efficiency inside the Pixel volume, as is expected. Conversely, the dark blue histogram shows only tracklets with exactly $N_{\text{pix}} = 3$, resulting in a significant efficiency between about 90 mm and the end of the Pixel detector. This corresponds to the radial range between the third and fourth Pixel layer. If a chargino decays between these two layers, it will have caused three hits in the innermost layers, but will not create one in the fourth. The tracklet reconstruction with reduced hit requirement is able to recover these hits, while the one with a requirement of $N_{\text{pix}} \geq 4$ is not.

Figure 8.3b shows a similar set of distributions. Here, the orange histogram represents an inclusive selection of tracklets with $N_{\text{pix}} \geq 3$, $N_{\text{SCT}} = 0$, including the SCT hit filter. The grey histogram inverts the SCT veto to $N_{\text{SCT}} > 0$. It still features only tracklets created by the second pass, and would have to be supplemented with standard tracks for a complete picture. It is clear, however, that using three- and four-layer tracklets results in a large reconstruction efficiency reaching from the third Pixel layer up to the SCT volume.

Also shown is the fraction of $\tilde{\chi}_1^{\pm}$ at a rest frame lifetime of $\tau(\tilde{\chi}_1^{\pm}) = 0.2$ ns that are expected not to have decayed at any given vertex radius, neglecting time dilation. The exact decay distance distribution therefore also depends on the mass and momentum of the chargino. It follows a classic exponential decay, and clearly shows the need to use dedicated short-tracklet reconstruction techniques, in order to achieve sensitivity at low lifetimes. By the time a chargino reaches the SCT volume, in order to possibly be picked up on by the standard reconstruction, it is overwhelmingly likely to already have decayed. Using these three-layer tracklets in addition to the four-layer ones also comes with certain drawbacks, two of which are discussed in the rest of this section.

8.2.1. Vertex constrained tracklet fit

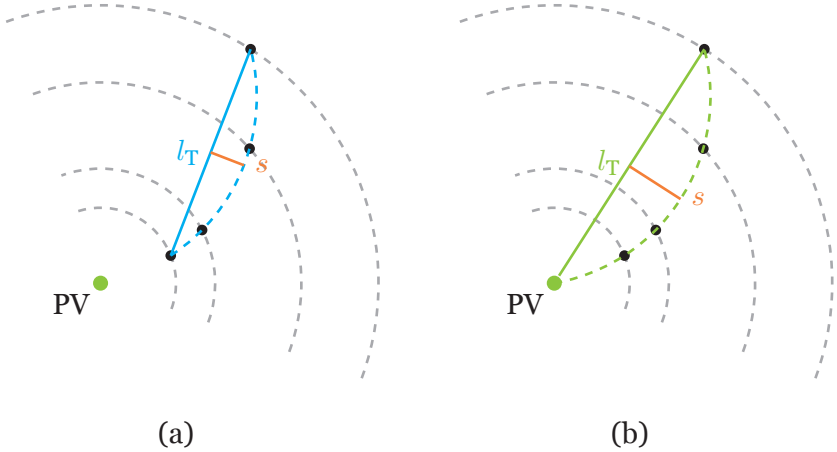
The first drawback of using short tracklets, especially the ones with only three hits is the significantly worse impact parameter and momentum resolution. The transverse momentum of a track is measured by interrogating the curvature of the trajectory induced by a magnetic field. If the magnetic field is known precisely, the accuracy of the measurement essentially depends on the length of the circle segment l_{T} of the trajectory projected onto the transverse plane like

$$\frac{\Delta p_{\text{T}}}{p_{\text{T}}} \propto \frac{1}{l_{\text{T}}^2}. \tag{8.2}$$

Clearly, short tracklets feature lower l_{T} than standard tracks, and therefore are expected to have a worse momentum resolution.

To improve this circumstance, the identified PV of the event can be supplemented as an additional measurement as described in [220]. This can either be achieved by rerunning the fitting routine and including the additional location in the minimization, or performing a reweighting of a linearized version of the trajectory after the fact. Both of these approaches result in a similar improvement of the overall resolution. Figure 8.4 illustrates how the

inclusion of the PV can increase the size of the track segment, in order to gain a better handle on the curvature of the trajectory.



(a) (b)

Figure 8.4.: Illustration of how including the PV in the trajectory fit can improve the p_T-resolution by increasing the segment length, thereby enhancing the fit's handle on the curvature. The blue line indicates the trajectory without the inclusion of the PV, while the green line does include it. The sagitta s associated with the segment is also drawn.

The improvement can be quantified by looking at the momentum distributions of simulated $\tilde{\chi}_1^\pm$ and associated tracklets that have been reconstructed from their charge deposits in the tracking system. The same simulated samples as the ones used to study the reconstruction efficiency are used. With the distribution of the relative discrepancy

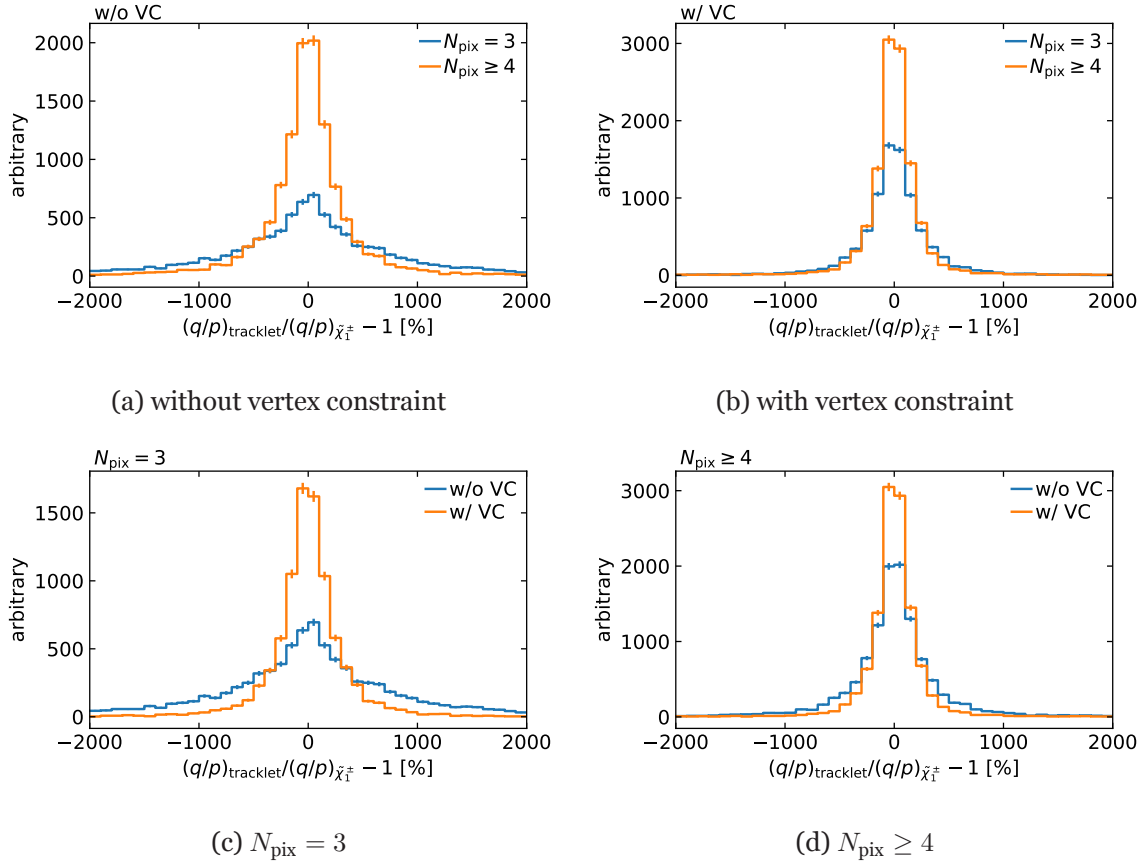
$$\frac{Q_{\text{tracklet}} - Q_{\tilde{\chi}_1^\pm}}{Q_{\tilde{\chi}_1^\pm}} \tag{8.3}$$

between the measured and real quantity Q, the impact of the vertex constrained refit can be quantified. Low values of this fraction indicate a closer match between the true and reconstructed value of the quantity Q. It can be calculated from tracklets matched to a true chargino particle in simulation.

Figure 8.5 shows normalized distributions of the relative discrepancy of q/p for the different tracklet types. q/p is shown instead of q/p_T, because it is the actual quantity being measured during the track fit via the curvature. As a consequence, the distribution should follow the expected bell-shape for a resolution curve.

Figure 8.5a shows the q/p resolution without the vertex constraint for three- and four-layer tracklets. With large widths and tails exceeding ± 1000 %, the plots clearly show that tracklet momentum resolution is very poor, a consequence of the short circle segment l_T. The width of the distribution is much broader for three-layer tracklets, indicating a worse reconstruction of the chargino momentum. For three-layer tracklets, the width at half maximum is around 140 %, while four-layer tracklets feature a much lower width or about 60 %. On the other

hand, figure 8.5b shows the equivalent distributions after the vertex constrained refit. Here, the difference in width is much smaller: The three-layer distribution has almost the same width as the four-layer one, at roughly 60 %. Clearly, the vertex constraint is able to recover a significant amount of precision for the shorter tracklets. Figures 8.5c and 8.5d shows equivalent distributions for three- and four-layer tracklets, where the resolution with and without the vertex constraint are overlaid.



(a) without vertex constraint

(b) with vertex constraint

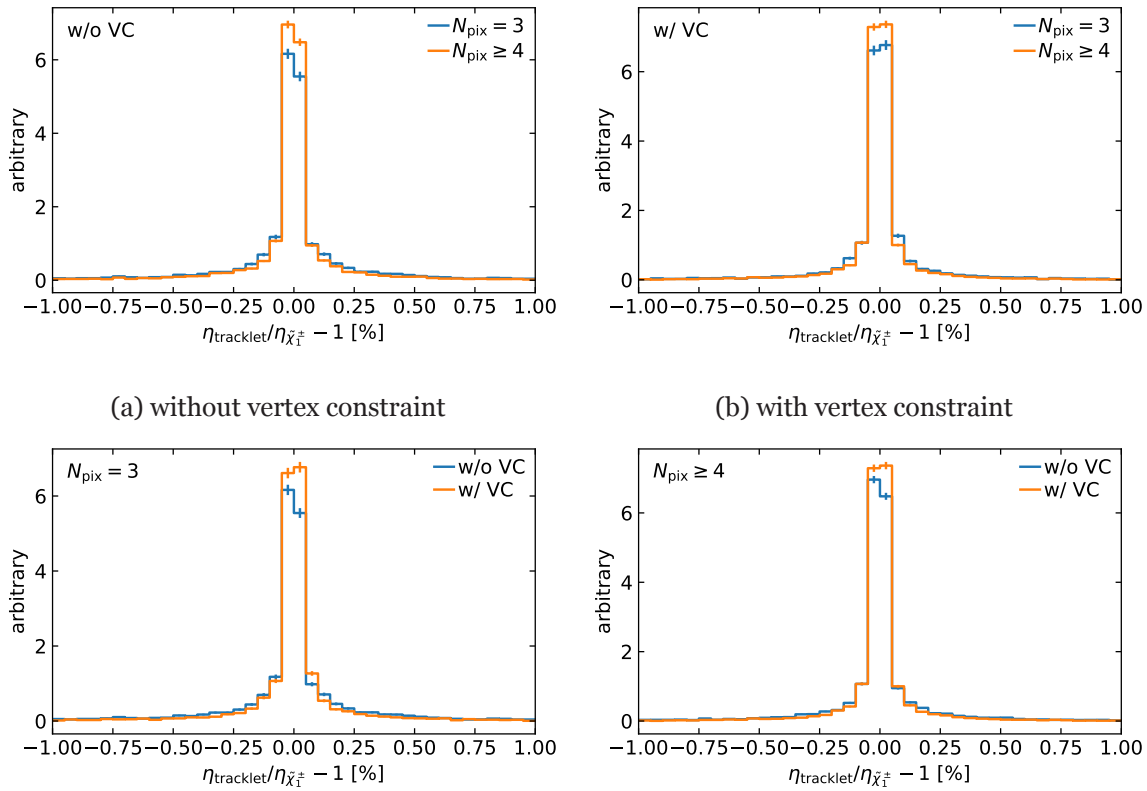
(c) $N_{\text{pix}} = 3$

(d) $N_{\text{pix}} \geq 4$

Figure 8.5.: Normalized distributions of the relative discrepancy between measured and $\tilde{\chi}_1^\pm$ q/p-value for three- and four-layer tracklets. The top row shows distributions with and without vertex constraint, and three- and four-layer tracklets are overlaid, while the bottom row shows the opposite.

Apart from the momentum measurement, the inclusion of the primary vertex in a refit also affects the determination of the of the angular quantities ϕ and θ of the resulting tracklet. Here, the different angular quantities paint a varied picture. Figure 8.6 shows the same normalized resolution distributions as seen before, but for η. As η is frequently used in analysis selections, it is shown here instead of θ. Figures 8.6a and 8.6b show the resolution with and without the vertex constraint, with the resolutions for three- and four-layer tracklets overlaid on top of each other. The distributions reveal almost no difference between the different tracklet types. At half of the maximum, the width is about 0.2 % in all cases, indicating that the vertex constraint does not have a large impact here. Figures 8.6c and 8.6d overlay resolutions with and without vertex constraint, and show similar compatible widths between the scenarios. This

circumstance can be explained by the fact that η corresponds to the angle in the longitudinal plane. In this plane, charged particle trajectories are not curved in the ATLAS ID, but instead form a straight line in the presence of a homogeneous magnetic field along the z-axis. The uncertainty to form a straight line is not significantly worse for three measurements compared to four measurements, therefore explaining the lack of difference in this variable.



(a) without vertex constraint

(b) with vertex constraint

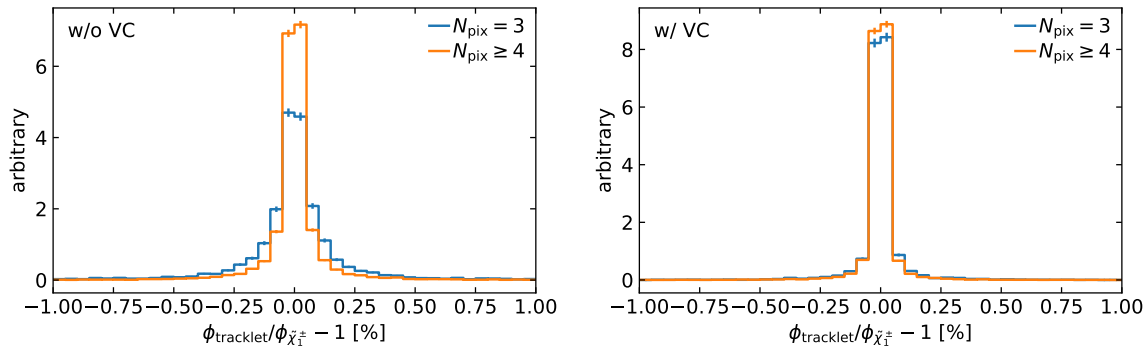
(c) $N_{\text{pix}} = 3$

(d) $N_{\text{pix}} \geq 4$

Figure 8.6.: Normalized distributions of the relative discrepancy between measured and $\tilde{\chi}_1^\pm$ η-value for three- and four-layer tracklets. The top row shows distributions with and without vertex constraint, and three- and four-layer tracklets are overlaid, while the bottom row shows the opposite.

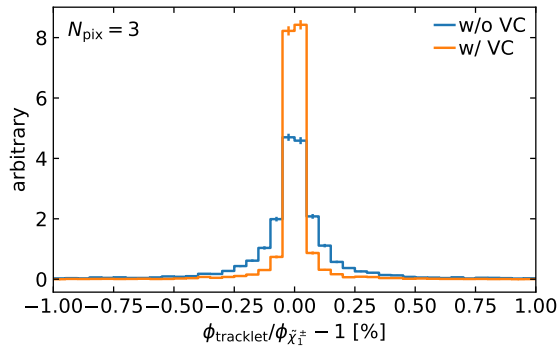
The transverse angular variable ϕ, on the other hand, is then expected to be affected to a similar degree as seen for the momentum variable q/p. Indeed, figure 8.7 shows this to be the case in the corresponding normalized resolution distributions. Here, again, the impact of the inclusion of the vertex constraint narrows the width of the resolution shape visible, from around 0.3 % to below 0.2 % (figure 8.7c) for three-layer tracklets, while its impact is negligible for four-layer tracklets (figure 8.7d). Figure 8.7b again reveals that the resolution is comparable between both tracklet types when the vertex constraint is used.

From the resolutions of the transverse quantities q/p and ϕ, it is apparent that the three-layer tracklets benefit to a larger degree from the additional information, since the distributions narrow considerably. However, four-layer tracklet resolutions are also improved. While momentum resolutions of this order of magnitude would be unacceptable for standard track

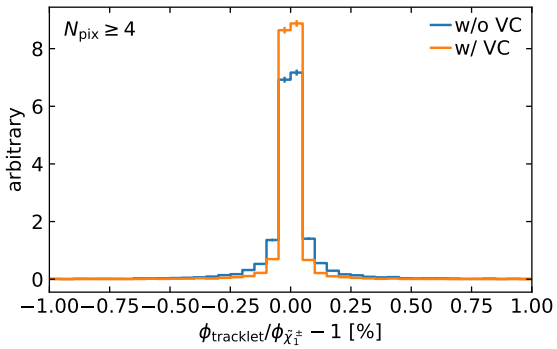


(a) without vertex constraint

(b) with vertex constraint



(c) $N_{\text{pix}} = 3$



(d) $N_{\text{pix}} \geq 4$

Figure 8.7.: Normalized distributions of the relative discrepancy between measured and $\tilde{\chi}_1^\pm$ ϕ-value for three- and four-layer tracklets. The top row shows distributions with and without vertex constraint, and three- and four-layer tracklets are overlaid, while the bottom row shows the opposite.

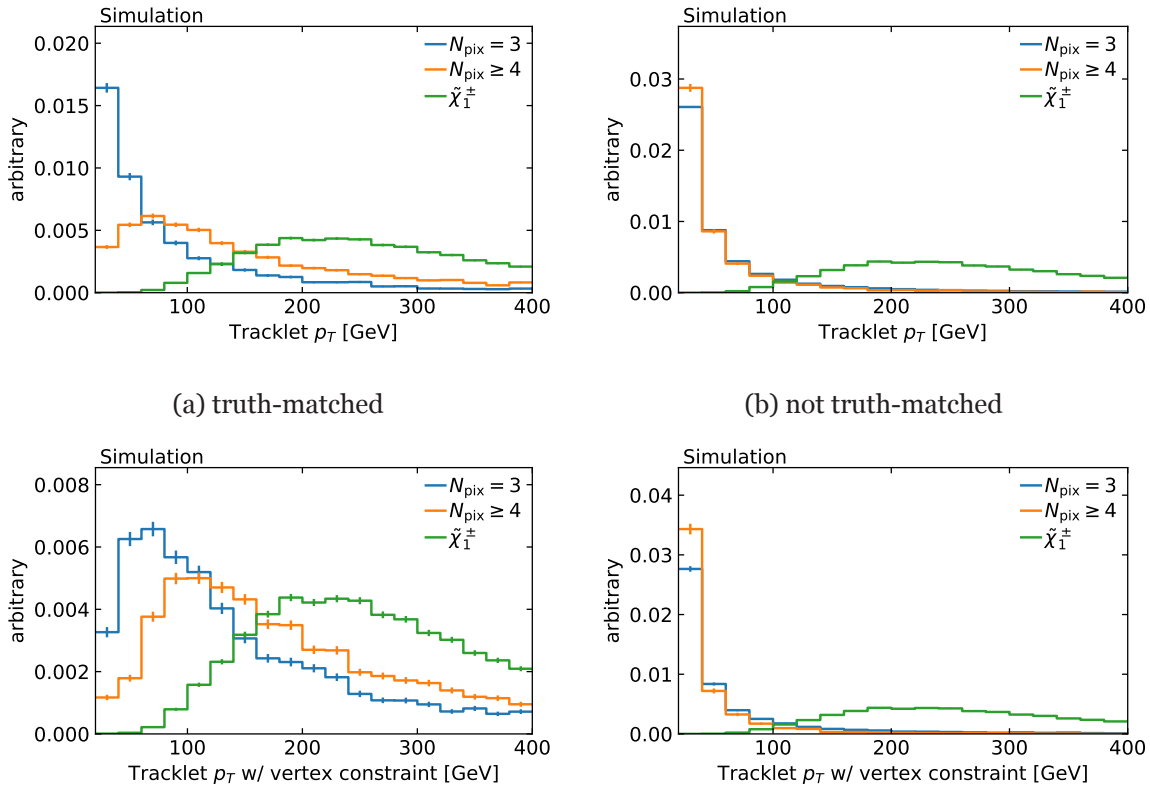
reconstruction and other types of analyses, with the vertex constraint, tracklet momentum resolution is deemed sufficient for a disappearing track analysis.

The analysis that is described in chapter 9 uses the vertex constrained values for p_{T}, η and ϕ where not otherwise indicated. While technically the impact parameters d_0 and z_0 could also be recalculated using the primary vertex constraint, these quantities are typically defined with respect to the primary vertex in the first place. This could introduce a bias, and the original uncorrected values are therefore used. Since cuts on the longitudinal impact parameter are often performed on $z_0 \sin\theta$, rather than the raw value, the vertex constrained value of θ enters the impact parameter selection directly to an extent.

8.2.2. Tracklet properties

With an understanding of how the tracklet reconstruction works and behaves, it makes sense to discuss the properties tracklets have in terms of various quantities. In the following, distributions are subdivided into truth-matched and not-truth-matched categories. The truth-matched tracklets are ensured to originate from a chargino decay, while the not-truth-matched

tracklets can either originate from other real particles producing tracks that disappear, or from random combinations of measurements. This manifests itself in different shapes of the distributions, and is discussed.



(a) truth-matched

(b) not truth-matched

(c) truth-matched

(d) not truth-matched

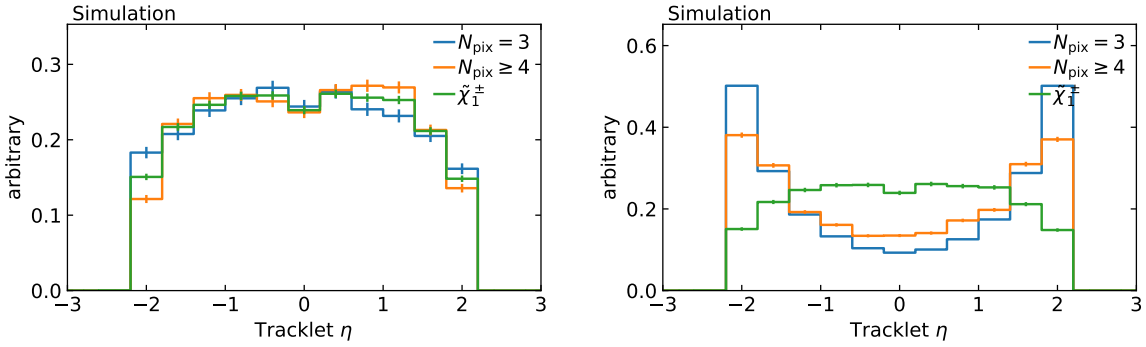
Figure 8.8.: Comparison of p_{T} values for reconstructed tracklets in $pp \to \tilde{\chi}_1^{\pm}\tilde{\chi}_{1,2}^{0}$ + X, $pp \to \tilde{\chi}_1^{\pm}\tilde{\chi}_1^{\pm} + X$ MC and true chargino momentum. The left side shows tracklets matched to a simulated chargino, while the right side shows tracklets that are not matched. Both p_{T} values with and without vertex constrained refit are shown. Distributions are scaled to unit area.

Figure 8.8 shows comparisons of p_{T} values of both types of tracklets. While figures 8.8a and 8.8b show raw p_{T} values, figures 8.8c and 8.8d contain values using the vertex constraint discussed in section 8.2.1. The distributions are normalized to unit area to allow comparing the shape, rather than the absolute numbers of tracklets. In addition, the true p_{T} spectrum of the chargino is shown as a reference.

The truth-matched tracklet distributions in the left column differ substantially between the regular p_{T} value and the vertex constrained one. In the former case, especially for three-layer tracklets, the chargino momentum is clearly underestimated, peaking at the very low edge of the histogram. For four-layer tracklets, the peak shifts to about 50 GeV, which is still considerably lower than the true momentum distribution. With the vertex constrained p_{T} shown in figure 8.8c, this improves to some degree. The three-layer distribution now peaks at about 40 GeV, while the four layer distribution gets pushed a little higher. Still, an

underestimation of the true momentum remains, which is consistent with the poor momentum resolution of these short tracks.

The right column of figure 8.8 shows tracklets which are not truth matched. As expected, the distributions are mostly concentrated in the lowest bins, while the raw p_{T} distribution presents a few more entries in the bins just above. Here, the inclusion of the vertex-constraint appears to pull the p_{T} toward lower values, there does not seem to be a qualitatively different behavior between the tracklet types.



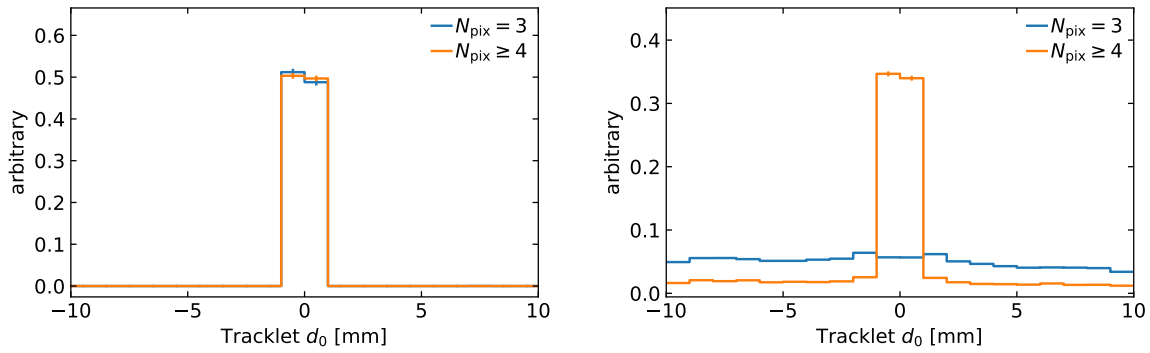
(a) truth-matched

(b) not truth-matched

Figure 8.9.: Comparison of pseudo-rapidity values of reconstructed tracklets in $pp \to \tilde{\chi}_1^{\pm}\tilde{\chi}_{1,2}^{0}$ + X, $pp \to \tilde{\chi}_1^{\pm}\tilde{\chi}_1^{\pm} + X$ MC. The left side shows tracklets matched to a simulated chargino, while the right side shows tracklets that are not matched. Distributions are scaled to unit area.

Distributions for the angular quantity η are shown in figure 8.9. Here, truth-matched tracklets shown in figure 8.9a follow the general shape as expected from the $\tilde{\chi}_1^{\pm}$ kinematics: a maximum at small values of $|\eta|$ which falls toward positive and negative values. Figure 8.9b shows not-truth-matched tracklets, and presents a very different shape. A minimum can be observed at $|\eta| \approx 0$, and the distributions rise toward larger positive and negative values. This shape can be explained by the larger likelihood to find random combinations of three or four silicon hits at larger absolute values of η, simply because the number of sensors that lie on a straight line outwards at that angle is larger. Random combinations are thus more likely to point away from the central region of the detector. No substantial differences can be observed between the two types of tracklets. In both figures, the chargino and seed level cut of $|\eta| < 2.2$ discussed previously can be observed.

As a next step, figure 8.10 shows distributions of the transverse impact parameter d_0. A large discrepancy between the truth-matched tracklets in figure 8.10a and the not-truth-matched ones found in figure 8.10b can be observed. While in the former, both three- and four-layer tracklets have a sharp maximum at low absolute values, and no population outside of the peak, the latter features different distributions for these two types. A maximum at low $|d_0|$ is expected for truth-matched tracklets, as the impact parameter is measured with respect to the primary vertex. In chargino decay events, the chargino production vertex is most likely to be the primary vertex, so an accurate reconstruction of the impact parameter will tend to



(a) truth-matched

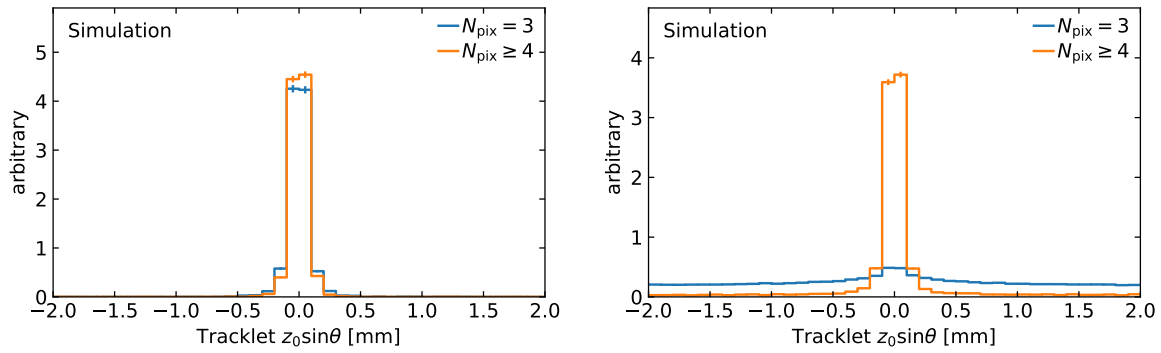
(b) not truth-matched

Figure 8.10.: Comparison of transverse impact-parameter values of reconstructed tracklets in $pp \to \tilde{\chi}_1^{\pm}\tilde{\chi}_{1,2}^{0}$ + X, $pp \to \tilde{\chi}_1^{\pm}\tilde{\chi}_1^{\pm} + X$ MC. The left side shows tracklets matched to a simulated chargino, while the right side shows tracklets that are not matched. Distributions are scaled to unit area.

zero. Tracklets not matched to charginos and with $N_{\text{pix}} \geq 4$ still peak at low absolute values of d_0, but show more or less symmetric slopes away from the peak. On the other hand, these tracklets with only $N_{\text{pix}} = 3$ do not peak at low values, but rather present a flat distribution across the entire spectrum of $-10\,\text{mm} < d_0 < 10\,\text{mm}$. This discrepancy can be explained by the fraction of random combinations in these populations. Purely combinatorial combinations of measurements do not prefer the interaction region as their origin, and thus are expected to have a uniform distribution of impact parameters. The fraction of these fakes is larger for $N_{\text{pix}} = 3$ than for $N_{\text{pix}} \geq 4$, simply because there is a larger pool of combinations. For the shorter tracklet type, this fake contribution is absolutely dominant, explaining the mostly flat distribution. For the longer ones, a population of real particles causing tracklets remains, and is found in the central region of the distribution.

Aside from this, it is also noticeable that the flat distribution of three-layer tracklets is not symmetric, but rather increases toward negative values. This might be caused by the track reconstruction attempting to fit a circle to three space points, an operation which always succeeds exactly, and could be biased toward lower values. Studies were performed to rule out problems with the charge identification, and the asymmetry induced by the tilted silicon sensors in barrel layers causing this effect.

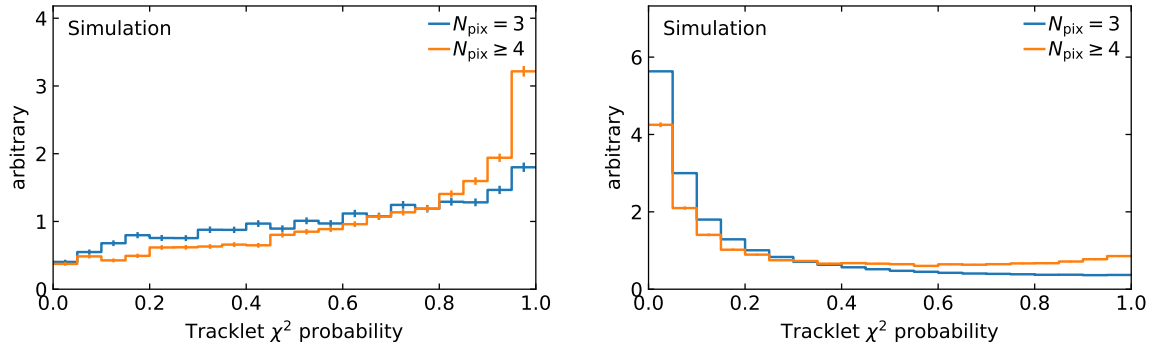
Similarly to d_0, the longitudinal impact parameter quantity $z_0 \sin\theta$ shown in figure 8.11 is also affected by the fraction of fake tracklets in the population. Figure 8.11a shows truth-matched tracklets, which again peak at low absolute values, and feature minimal tails. This is consistent with tracklets following the property of the underlying matched $\tilde{\chi}_1^{\pm}$ particle. Conversely, not-truth-matched tracklets in figure 8.11b, differ in shape by their type. The longer tracklets again peak at zero, meaning that they are associated with real particles originating in the vicinity of the primary vertex, and presenting a disappearing track. A non-zero population of larger values, that is mostly flat can also be found, and can be identified with the population of fake tracklets with $N_{\text{pix}} \geq 4$. As before, since the combinatorics are



(a) truth-matched (b) not truth-matched

Figure 8.11.: Comparison of longitudinal impact-parameter values of reconstructed tracklets in $pp \to \tilde{\chi}_1^{\pm}\tilde{\chi}_{1,2}^0$ + X, $pp \to \tilde{\chi}_1^{\pm}\tilde{\chi}_1^{\pm} + X$ MC. The left side shows tracklets matched to a simulated chargino, while the right side shows tracklets that are not matched. Distributions are scaled to unit area.

larger, the shorter tracklet type with $N_{\text{pix}} = 3$ has a larger population of fake tracklets, and is therefore mostly flat in $z_0 \sin\theta$. A small peak at zero can be found however, hinting at a real contribution to this type of tracklet as well.



(a) truth-matched (b) not truth-matched

Figure 8.12.: Comparison of fit quality values of reconstructed tracklets in $pp \to \tilde{\chi}_1^{\pm}\tilde{\chi}_{1,2}^0$ + X, $pp \to \tilde{\chi}_1^{\pm}\tilde{\chi}_1^{\pm} + X$ MC. The left side shows tracklets matched to a simulated chargino, while the right side shows tracklets that are not matched. Distributions are scaled to unit area.

Finally, figure 8.12 shows the distribution of χ^2-probabilities of the tracklets, which is a property that can be extracted from the trajectory fit procedure. This quantity can be used to quantify the quality of the parameter fit to the available measurements. Generally, a good fit features a χ^2-value that follows a χ^2-distribution. The probability shown here corresponds to the p-value of any given value of χ^2 in the assumed associated χ^2-distribution for a specific number of degrees of freedom. A population of good fits would then result in a uniform picture of χ^2-probabilities, as this implies that the individual χ^2-values do indeed follow the

correct distribution.

Looking at figure 8.12a it is obvious that this is clearly not the case: both types of truth-matched tracklets tend toward larger values of the χ^2-probability, which does not indicate a better fit quality. Especially the shorter tracklets show a relatively flat region between about 0.2 and 0.8, however. This is counter-intuitive, as a fit with a larger number of measurements and thus more information would be assumed to be better. An explanation can be found by reasoning about what the fit actually does. For short tracklets, the algorithm will essentially attempt to fit a circle to three space points in the transverse plane. This always succeeds, it is simply a matter of finding the parameters to build a circle that passes through all three points. The minimization of the χ^2-value is then effectively simply looking for the parameters which yield $\chi^2 = 0$. For more spacepoints, the algorithm has to perform an optimization to find the lowest χ^2, which results in a worse fit-quality in terms of the quantity discussed here.

Nevertheless, the χ^2-probability is a useful property, which becomes clear when looking at the not-truth-matched tracklets in figure 8.12b. In contrast to the tracklets belonging to charginos, the distribution tends toward zero, while again showing a clear flat baseline. It therefore presents a good discriminant to select tracklets originating from chargino decays.

9. Search for long-lived charginos in the ATLAS detector

9.1. Overview and analysis strategy

Chapter 9 describes a search for Supersymmetry (SUSY), in a scenario where the LSP is a pure higgsino. A discussion of the characteristics of this signal model is given in section 9.2. The signal model features a long-lived pure-higgsino charged gaugino, which decays while traversing the ATLAS detector. Due to its lifetime, this particle produces a disappearing track signature, that was the focus of chapter 8. Dedicated reconstruction techniques for very short tracks, called *tracklets*, are used.

An introduction of various background processes, that can result in signatures similar to the signal model, is given in section 9.3. The main backgrounds are scattered electrons, muons or hadrons, as well as random combinations of measurements that are reconstructed as tracklets.

The dataset which is used to perform the search for this signal process was obtained with the ATLAS experiment at the LHC in 2018, during the *Run 2* proton-proton data-taking campaign, at $\sqrt{s} = 13\,\text{TeV}$. Details on how the data was recorded and preprocessed are given in section 9.4. In the same section, a description of the simulated MC samples used to quantify the signal expectation is given.

A selection procedure is defined, in order to separate signal process events from background. At the event level, significant missing transverse energy is required, as is a high-p_{T} jet. Definitions and preconditions for the set of physics objects used in the event selection are given in section 9.5. Here, information on the resolution of double-counting between the objects, and details on the calculation of the $E_{\text{T}}^{\text{miss}}$ variable can be found as well. Finally, section 9.6 describes in detail how the selected objects are combined to form event-level criteria, in addition to other requirements.

To quantify compatibility of observed data with the signal and background hypotheses, the expected background has to be estimated. As the details of the tracklet reconstruction, which is the centerpiece of the analysis, are very difficult to accurately simulate, data-driven methods are used. Two approaches for the estimation of the background are presented. One approach makes a prediction of the contribution of different sources of background, and is shown in section 9.7. It uses different techniques to obtain estimates of the shape of the background transverse momentum distribution, broken down by different processes. These shape estimates are called *templates*. Scattered leptons and hadrons are estimated using a *tag-and-probe* approach, that results in a transfer factor from a single-lepton control region

into the signal region. With these transfer-factors, templates for these background sources are derived. Templates for the events containing fake tracklets are derived from a tracklet quality sideband, which is enriched in fakes with respect to the signal region. The absolute normalization of these templates is derived in a statistical likelihood fit. Another approach which instead uses a unified estimation method based on an ABCD method is presented in section 9.8. This method does not predict the exact background composition, but gives an estimate of the overall background normalization instead.

Finally, a binned likelihood fit is carried out to perform a statistical analysis of the dataset, using the inclusive ABCD background estimation in combination with the signal expectation. A study of systematic uncertainties affecting both the simulation based signal expectation as well as the data-driven background estimate is presented. With a pseudo-data set in the signal region, the expected sensitivity is derived in the form of signal strength upper limits. The impact of recent developments in tracklet reconstruction is quantified.

Comparisons to existing publications are made, and future improvements to the analysis are discussed. In addition, the prospects of searches of this kind with future colliders are studied briefly.

9.2. Signal model

9.2.1. Aspects and characteristics

Following the discussion in sections 2.3 and 2.3.1, the signal model is situated within the Minimal Supersymmetric Standard Model (MSSM) assuming Anomaly Mediated Supersymmetry Breaking (AMSB). Here, two supermultiplets group the SM Higgs field, in addition to three new scalar fields. The four associated fermion superpartners are called *higgsinos*, and mix with the superpartners of the electroweak bosons before electroweak symmetry breaking.

As contemporary LHC searches place stringent limits on large portions of the SUSY particle spectrum, model varieties which can explain the lack of discoveries are becoming more popular. In the scenario of split mass sectors, the majority of supersymmetric particles can conveniently be pushed beyond current detection capabilities. The mass spectrum of the lower sector is then characterized by the couplings of the neutralinos and charginos. More specifically, the low-mass sector particles $\tilde{\chi}_1^{\pm}$ and $\tilde{\chi}_{1,2}^0$ are almost degenerate, with only a small mass-splitting

$$\Delta m_+ = m_{\tilde{\chi}^{\pm}} - m_{\tilde{\chi}^0} = \Delta m_{\text{tree}} + \Delta m_{\text{rad}} \tag{9.1}$$

that is the sum of a small tree-level contribution, while the rest is caused by radiative effects. Depending on the model scenario, Δm_{tree} is either exactly zero (pure-higgsino) or up to about 100 MeV. Even in the pure-higgsino case, Δm_{rad} can be 100 MeV or larger. The dominant decay channel of the $\tilde{\chi}_1^{\pm}$ is to a charged pion, if the mass difference Δm_+ permits it. The

partial width for this decay is

$$\Gamma(\tilde{\chi}_1^\pm \to \chi_{1,2}^0 \pi^\pm) = \frac{G_F^2}{\pi} \cos^2\theta_c f_\pi^2 \Delta m_+^3 \sqrt{1 - \frac{m_\pi^2}{\Delta m_+^2}}, \tag{9.2}$$

where G_F is the Fermi constant, θ_c the Cabibbo angle, $f_\pi \simeq 130\,\text{MeV}$ the pion decay constant, and m_π the pion mass. Another possible decay is to (anti-)leptons and associated (anti-)neutrinos: $\tilde{\chi}_1^\pm \to \tilde{\chi}_{1,2}^0 l^\pm [\nu/\bar{\nu}]$. The branching ratio to pions is over 95.5 % for the mass ranges of interest [82].

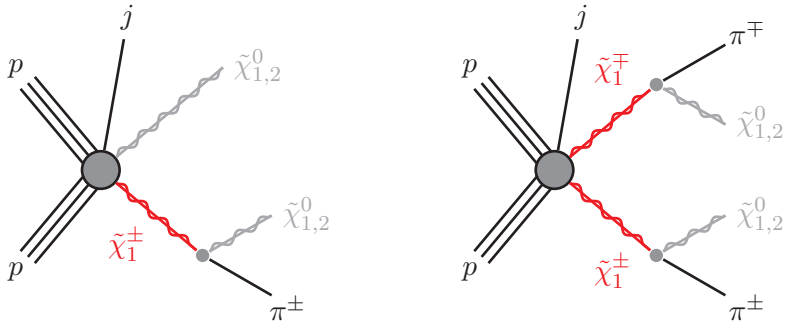


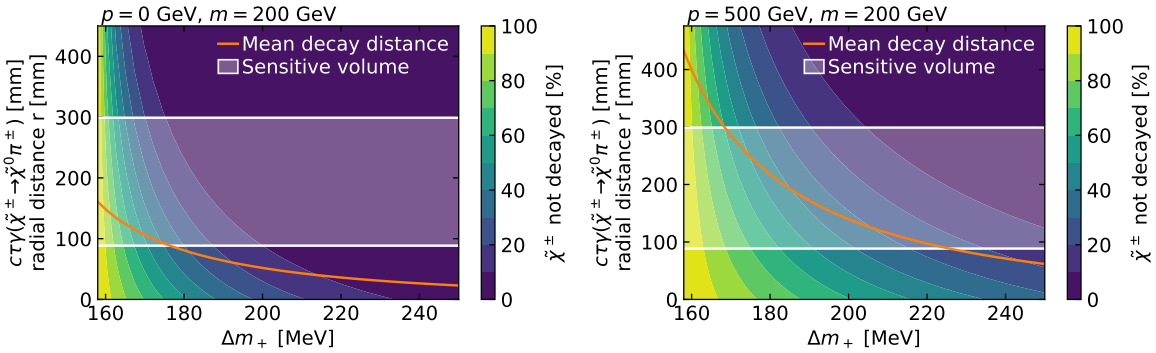
Figure 9.1.: Feynman diagrams showing the production and decay of charginos in proton-proton collisions. An ISR jet is drawn as well.

Figure 9.1 shows Feynman diagrams of the relevant processes where either a pair of $\tilde{\chi}_1^+\tilde{\chi}_1^-$ or $\tilde{\chi}_1^\pm\tilde{\chi}_{1,2}^0$ is produced. Subsequently, the charginos decay into a $\pi^\pm$ and a $\tilde{\chi}_{1,2}^0$. Also shown is an Initial State Radiation (ISR) recoil jet, which is required to provide the chargino with enough transverse momentum. Using equation (9.2), the travel distance of the chargino before decaying can be calculated [221] via $\tau = 1/\Gamma$ as

$$c\tau(\tilde{\chi}_1^\pm \to \tilde{\chi}_{1,2}^0 \pi^\pm) = 1.1\,\text{cm} \left(\frac{\Delta m_+}{300\,\text{MeV}}\right)^{-3} \left[1 - \frac{m_\pi^2}{\Delta m_+^2}\right]^{-1/2}. \tag{9.3}$$

Note that this calculation is an approximation, as it only considers the decay mode to a pion, and neglects leptonic decays. The mean decay distance $c\tau$ from equation (9.3) is valid in the chargino rest frame, and needs to be corrected with the Lorentz factor γ to account for time dilation. The relationship of lifetime and mass splitting is visualized in figure 9.2a, where the mean travel distance $c\tau\gamma$ ($\gamma = 1$), is drawn as a function of Δm_+ in orange. A white shaded area indicates the radial extent of the sensitive volume between the third layer of the ATLAS Pixel detector at $r = 88.5\,\text{mm}$, and the first layer of the SCT at $r = 299\,\text{mm}$ (see figure 8.3b). The contour lines give an indication of the remaining fraction of charginos at any point $(\Delta m_+, r)$, based on the mean decay distance $c\tau\gamma$ for that Δm_+. Especially for $\Delta m_+ \lesssim 200\,\text{MeV}$, a significant portion of charginos actually travel far enough to reach the covered range. In that case, there is a chance of being able to reconstruct the chargino.

In the signal selection, a jet is required in the event. If this jet is an ISR jet, it gives transverse



(a) $p = 0$ GeV (b) $p = 500$ GeV

Figure 9.2.: Chargino mean travel distance $c\tau\gamma$, as a function of mass-splitting Δm_+ between chargino and neutralino in orange. Contour lines give an indication of the remaining fraction of charginos at any point $(\Delta m_+, r)$, based on the mean decay distance $c\tau\gamma$ for that Δm_+. The radial extent of the sensitive volume between $r = 88.5$ mm and 299 mm is marked. Two different chargino momenta and the effect of time dilation are displayed.

momentum to the chargino. Depending on the chargino momentum, its decay length will be enhanced by relativistic time dilation, resulting in an increased decay distance in the laboratory frame. This effect can be seen in figure 9.2b which shows the mean decay distance and fraction of charginos that have not decayed yet at a given radial distance, for a momentum of 500 GeV as a function of Δm_+. The range of mass splittings for which the mean decay distance is inside the sensitive volume is larger than in figure 9.2a.

Nevertheless, this effect does not uniformly increase the fraction of reconstructible charginos. While more charginos enter the sensitive volume between the third Pixel and first SCT layer, an increased number of charginos also enters the SCT before decaying. Charginos leaving measurements in the SCT will be rejected in the event selection. The maximum fraction of charginos decaying in the sensitive volume depends on the mass, momentum and mean lifetime. Figure 9.3 shows this fraction as a function of chargino momentum for $\tau = 0.2$ ns and three masses. At this lifetime, the fraction is maximized at about 400 GeV to 600 GeV.

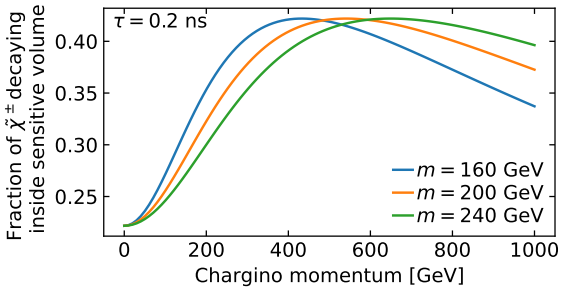
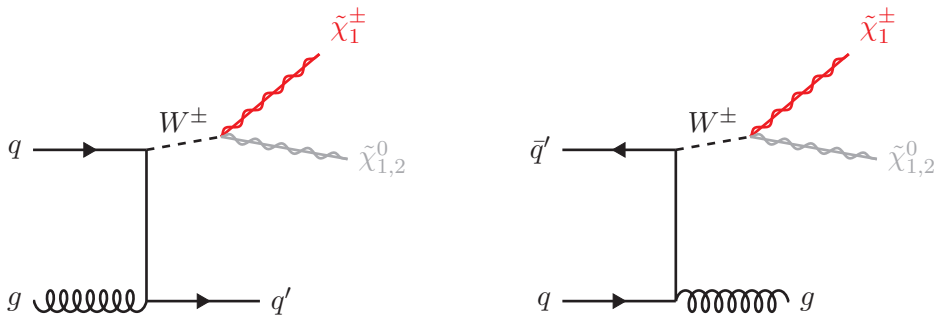
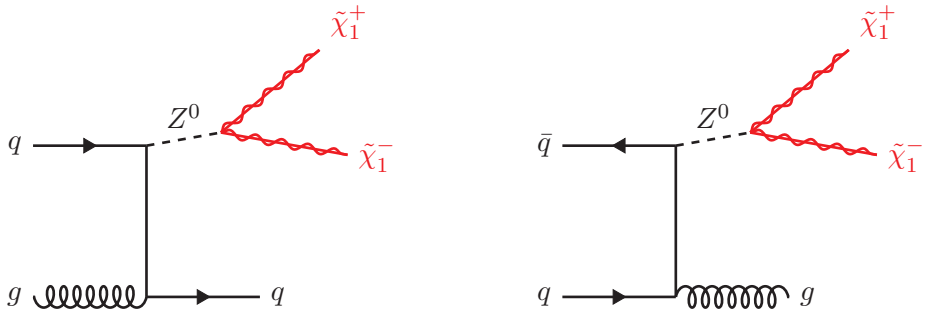


Figure 9.3.: Fraction of charginos decaying inside the sensitive volume as a function of chargino momentum. The fraction is given for $\tau = 0.2$ ns and different masses.

Eventually, the produced neutralino will inherit most of the chargino's momentum. Since the neutralino is the LSP in this signal model, and R-parity is assumed to be conserved, it will carry this momentum out of the detector without being reconstructed. The other decay product, the pion, is typically too soft to be picked up by standard reconstruction algorithms (see section 9.11.2). This manifests itself in an imbalance of the overall p_{T} of the event, which is reconstructed as missing transverse energy $E_{\mathrm{T}}^{\mathrm{miss}}$ (section 3.5.4). A large value of $E_{\mathrm{T}}^{\mathrm{miss}}$ is also used as a trigger requirement due to the characteristics of the signal process



(a) $pp \to j\tilde{\chi}_1^{\pm}\tilde{\chi}_{1,2}^0 + X$



(b) $pp \to j\tilde{\chi}_1^{\pm}\tilde{\chi}_1^{\mp} + X$

Figure 9.4.: Feynman diagrams of typical production processes resulting in either a pair of $\tilde{\chi}_1^{+}\tilde{\chi}^{-}$ (a) or $\tilde{\chi}_1^{\pm}\tilde{\chi}_{1,2}^0$ (b), in the presence of a jet emitted from the initial state.

Figure 9.4 shows diagrams of production processes $pp \to j\tilde{\chi}_1^{\pm}\tilde{\chi}_1^{\mp} + X$ and $pp \to j\tilde{\chi}^{\pm}\tilde{\chi}_{1,2}^0 + X$. The initial state consists of two incoming partons, either a pair of quarks or a combination of a quark and a gluon. In both cases, a quark or a gluon is emitted as an ISR jet. Having undergone the jet emission, the parton will interact with the remaining initial state parton to form either a neutral Z^0 or a charged $W^{\pm}$, which subsequently decays into the final state charginos and neutralinos. The ratio between the production cross-sections is about 2:7 between $\tilde{\chi}_1^{\pm}\tilde{\chi}_1^{\mp}$ and $\tilde{\chi}_1^{\pm}\tilde{\chi}_{1,2}^0$ [222].

9.2.2. Existing results of searches

Building on top of LHC Run 1 disappearing track searches by ATLAS [218, 223] and CMS [224] at $\sqrt{s} = 7\,\mathrm{TeV}$ and 8 TeV, respectively, both experiments have published results [219, 225] with partial Run 2 data. In these analyses, the Run 1 lower mass bounds on long-lived

charginos were improved from 270 GeV and 260 GeV at $\tau = 0.2$ ns to 460 GeV and about 400 GeV at 0.2 ns and 0.3 ns, respectively.

Experimental searches for pure-higgsino scenarios with low Δm_+ using disappearing tracks have only recently started to be conducted. In case of larger Δm_+, other signatures such as opposite-sign lepton pairs and $E_{\mathrm{T}}^{\mathrm{miss}}$ are more sensitive [226–228] ($\Delta m_+ \simeq 20$ GeV). A reinterpretation of the ATLAS pure-wino analysis to a pure-higgsino model was published [229], providing exclusion of masses lower than 152 GeV at the nominal lifetime predicted by theory.

CMS published an analysis using the full Run 2 dataset [222], featuring interpretations in the context of both pure-wino and pure-higgsino signal scenarios. The pure-wino signal scenario is excluded for masses below 474 GeV at 0.2 ns, while masses below 175 GeV are excluded for the pure-higgsino signal, evaluated at $\tau = 0.05$ ns.

These analyses, or the analyses they are based on, are primarily designed for the pure-wino scenario, and only slightly adjusted for pure-higgsino interpretations. Therefore, they perform best at comparatively larger lifetimes. The analysis presented in this chapter is designed for the pure-higgsino scenario, and therefore uses very short tracklets. This has the goal of improving sensitivity in that parameter space.

9.3. Background sources

In order to conduct a search for disappearing tracks caused by the decay of a pure-higgsino $\tilde{\chi}_1^{\pm}$, it is critical to understand other processes that can result in signatures that are the same or similar to the signal. At the event level, the signature is characterized by a high-p_{T} jet and approximately equally large $\vec{E}_{\mathrm{T}}^{\mathrm{miss}}$, which is roughly pointing in the opposite direction. The dominant process resulting in such a signature is the production of $W^{\pm}$-bosons with a subsequent decay to a charged lepton $l^{\pm}$ and a corresponding neutrino $\nu_l/\bar{\nu}_l$, in combination with a jet, as shown in figure 9.5. Here, an example of the process in association with a gluon jet is shown. As the $W^{\pm}$ produces a neutrino upon decay, it results in a certain amount of $E_{\mathrm{T}}^{\mathrm{miss}}$, since the neutrino will not be detected.

Aside from these event-level quantities, the signal signature also requires a disappearing track. Multiple possibilities for how such a track can appear in the event are discussed in the following.

9.3.1. Charged lepton scatter background

Charged particles can undergo scattering while traversing the material that makes up a particle detector. This effect is governed by properties of the particle, like its momentum, and of the material properties. It becomes relevant for the processes $W^{\pm} \to e^{\pm}[\nu_e/\bar{\nu}_e]$ and $W^{\pm} \to \mu^{\pm}[\nu_\mu/\bar{\nu}_\mu]$ in the presence of a recoil jet, where the final state $e^{\pm}$ or $\mu^{\pm}$ traverses the detector and possibly scatters. Usually, the scattering angles are reasonably small, and therefore can be resolved by the track reconstruction. A track fit can accommodate a trajectory

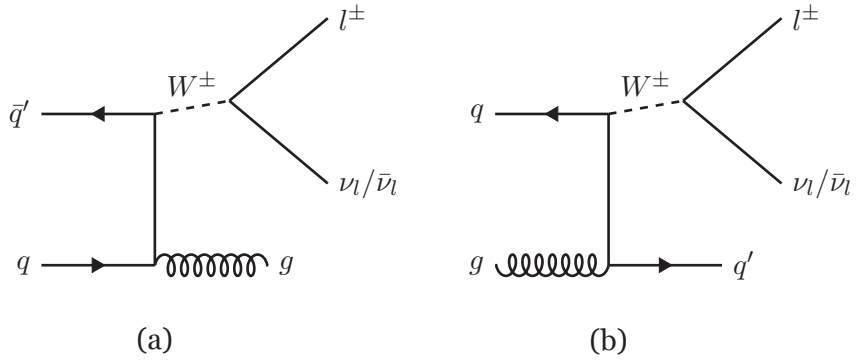


Figure 9.5.: Example Feynman diagrams of a $W^{\pm}$-boson being produced alongside a jet. The $W^{\pm}$ subsequently decays into a charged (anti-)lepton $l^{\pm}$ and the corresponding (anti-)neutrino $\nu_l/\bar{\nu}_l$.

that changes direction at some point to certain extent. However, this usually results in a worse estimate of the parameters.

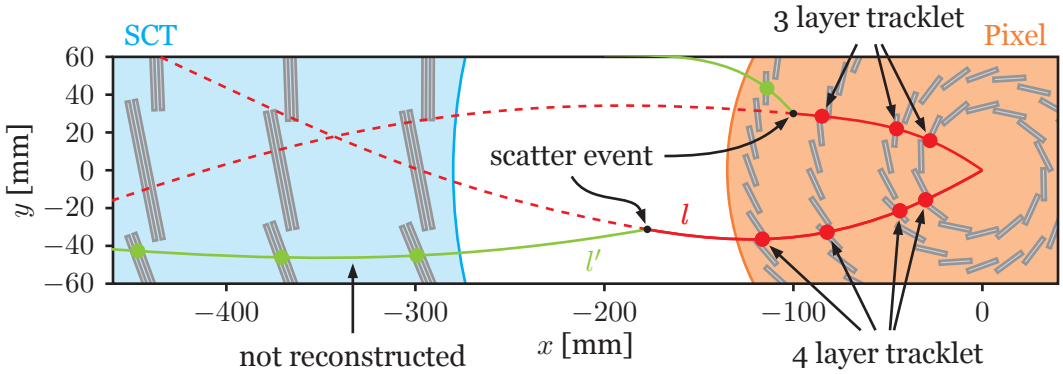


Figure 9.6.: Illustration of how a scattered lepton can produce a signature the resembles a disappearing track. As an example, two leptons are emitted from the interaction region, and scatter after having traversed parts of the Pixel detector volume. Curvatures are not to scale.

However, if a charged particle scatters severely enough, the standard reconstruction might not be able to pick up its signature. Such a kinked track could then either result in a failure to reconstruct the particle entirely, or produce a trajectory consisting partially of the particle's hits as well as mismatched measurements.

This effect is illustrated in figure 9.6. Here, a pair of leptons[1] is produced in the interaction region of the detector and traverses the Pixel detector volume, while depositing energy in the sensors. One of the particles scatters before reaching the fourth layer. The scattered lepton ends up also creating a hit in the fourth layer. However, it will deviate from the expected intersection of the trajectory and the layer by so much, that the reconstruction does not associate it correctly. The other lepton scatters between the Pixel and the SCT volumes.

[1]The probability of both leptons scattering so severely in the same event is exceedingly low. This topology is chosen for illustrative purposes.

Further hits are created in the SCT, but they are again not close enough to be associated with the rest of the trajectory, by the standard reconstruction.

Both of these cases would result in a Pixel tracklet, as the full trajectory is not found by the standard reconstruction. The tracklet reconstruction is sensitive to the short track created before scattering. The former case would result in a three-layer tracklet. However, this case is exceedingly unlikely, since the amount of material found between the third and the fourth layer is minimal, and the distance available for the scattering is also small. As the gap between the Pixel and SCT volumes is larger and contains more material, it is more likely to contain scattering events, reconstructed as four-layer tracklets.

9.3.2. Hadron scatter background

Aside from the production of electrons and muons, the $W^{\pm}$ in the process discussed previously can also decay into a τ-lepton and neutrino. These τ-leptons have a high probability to decay hadronically, mostly producing charged and neutral pions in the final state in addition to $E_{\mathrm{T}}^{\mathrm{miss}}$.

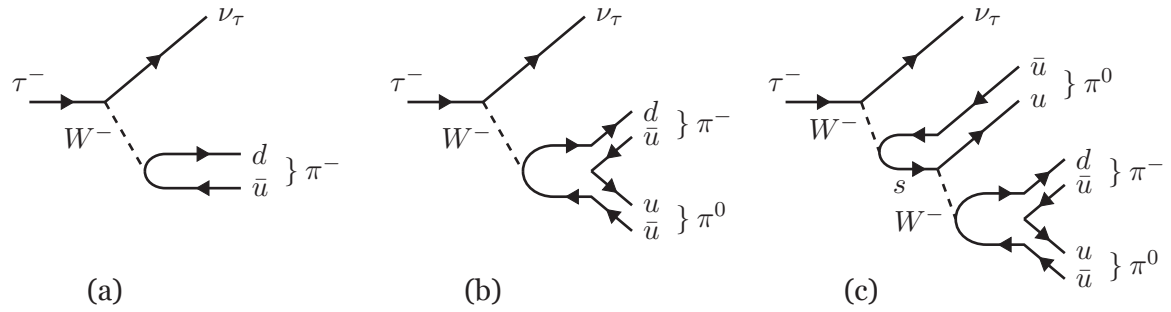


Figure 9.7.: Feynman diagrams of the dominant hadronic τ-lepton decay modes into one, two or three pions. Examples of negative τ^- decaying are shown, but analogous τ^+ diagrams exist as well.

Three dominant hadronic decay modes of the τ lepton are shown in figure 9.7: $\tau^{\pm} \to [\bar{\nu}_\tau/\nu_\tau]\pi^{\pm}$, $\tau^{\pm} \to [\bar{\nu}_\tau/\nu_\tau]\pi^{\pm}\pi^0$ and $\tau^{\pm} \to [\bar{\nu}_\tau/\nu_\tau]\pi^{\pm}2\pi^0$. These three processes make up about 46 % of all τ decays [46]. Like electrons and muons, the charged pions can undergo scattering in the detector, thereby producing a tracklet, that can enter the signal selection. As is the case for charged lepton scatters, this type of background is expected to be much more relevant for four-layer tracklets than for three-layer tracklets, for the same material density and geometric reasons.

9.3.3. Combinatorial fake background

Finally, there is a source of tracklet candidates which does not originate from scattered leptons nor hadrons. Due to the reduced amount of hits that are required to form a tracklet, there is a possibility of reconstructing random combinations of hits. In the seeding phase, triplets of hits are assembled and checked for compatibility with the interaction region, and large

enough transverse momentum. If a random triplet combination fulfills these requirements by chance, it is passed on to the track finding and fitting stage. Even though requirements placed on the distance hits can have from the extrapolated trajectory, random tracklets can still pass these stages. These types of tracklets are referred to as *fakes*, due to them not originating from an actual particle. If an event passes the large transverse momentum jet and large $E_{\mathrm{T}}^{\mathrm{miss}}$ requirements, it can potentially pick up a fake tracklet and enter the signal selection.

The prevalence of this type of tracklets depends on the density of hits in the eligible layers of the detector: The number of combinations increases with the number of hits. As a consequence, this type of background is expected to be highly pile-up dependent, since additional interactions are likely to generate more hits in the Pixel layers.

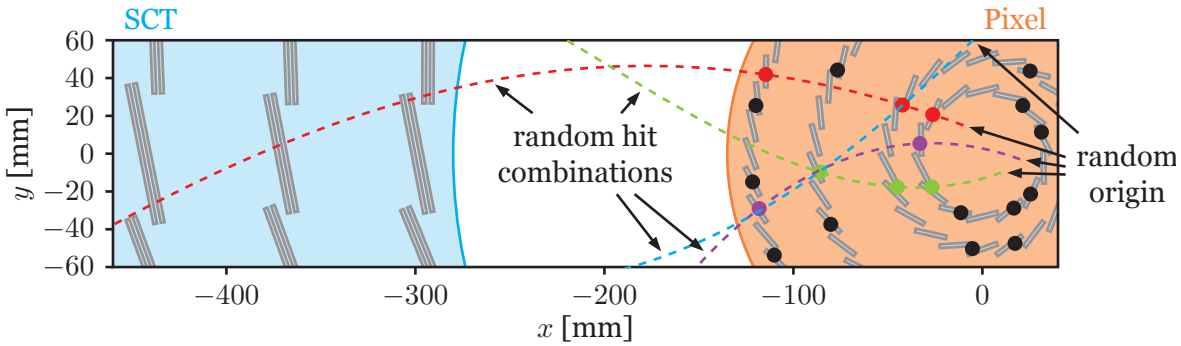


Figure 9.8.: Illustration of the formation of *fake* tracklets from random combinations of hits in the Pixel detector. Four random combinations are shown connecting various hits. Their origins do not strictly point toward the interaction region.

An illustration of the construction of the fake background from random combinations of hits can be found in figure 9.8. Here, a number of random hits are drawn on the Pixel layers. Four exemplary trajectories connect subsets of these hits in a random fashion. Some of them share hits, although this assignment ambiguity between hits and tracks is resolved during the reconstruction phase. Note that the trajectories formed from random combinations do not favor pointing toward the interaction region, as is the case for tracklets originating from scattered particles.

This manifests itself in the distribution of the longitudinal and transverse impact parameters with respect to the PV: as there is no preference for low impact parameters, these distributions are expected to be more or less uniform. In contrast to this, impact parameters of tracklets originating from the PV peak at low absolute values.

9.4. Datasets and simulated samples

9.4.1. Recorded dataset

The analysis presented in this chapter uses data collected with the ATLAS experiment (see chapter 3) at CERN. From 2015 to 2018, the LHC conducted its second data-taking phase

named Run 2, with an increased center-of-mass energy of $\sqrt{s} = 13\,\text{TeV}$. In total, proton-proton collisions amounting to $147\,\text{fb}^{-1}$ of integrated luminosity were recorded. However, when analyzing such datasets, the requirement that all parts of the detector were operating without issues is demanded in addition. This reduces the total amount of integrated luminosity that is deemed suitable for analysis to $139\,\text{fb}^{-1}$. A centralized process is applied to the dataset: After recording, the data is converted from the RAW readout format into a higher-level representation. Subsequently, reconstruction algorithms (see section 3.5) are executed on it, which turns the electronic signals that were recorded into low-level signatures, such as tracks or calorimeter clusters. Dedicated algorithms tuned to the various details of each domain use these low-level signatures to produce physics objects. For each type, the objects are refined to provide calibrated values for measured quantities, as well as providing information on how certain the algorithms are that the low-level signature actually belong to a hypothesized physics object. Finally, the data is written out in an analysis-friendly way, and further filtered to produce various dedicated analysis formats. At this level, additional algorithms calculate auxiliary values and quantities, that serve the selection and the target analyses of the format.

The disappearing track signature used in this analysis can only be reconstructed by specialized algorithms (see section 8.2). These reconstruction algorithms run in a second pass after the standard track reconstruction. Even though they only work on measurements that have not been used by the first pass, this second pass is computationally expensive due to the increased combinatorics. This is a direct consequence of the relaxed minimum hit requirement.

To maintain a reasonable processing time, the algorithms are not run as part of the default reconstruction procedure. Instead, the `DRAW_RPVLL` [230] filter is used. This filter executes on all RAW events, runs the configured reconstruction, applies a selection, and finally stores the events which pass the filter criteria. The main `DRAW_RPVLL` filter consists of several subfilters, each tuned toward specific signatures and analyses. The *KinkTrk* filter, which is geared toward the disappearing track analysis, selects events with a certain minimum $E_{\text{T}}^{\text{miss}}$, a jet with a minimum p_{T} and a successfully reconstructed tracklet. Additionally, it contains a configuration to select $Z^0 \to e^+e^-$ and $Z^0 \to \mu^+\mu^-$ events, which are required for the background estimation procedure (see section 9.7.2). To keep the output filter rate reasonably low, the Z^0 filter streams are prescaled by a factor of five, which means that only every fifth event that is accepted by the filter is actually kept.

The algorithms to produce the data formats used in analyses are run again only on these altered DRAW events. If subsequent changes are made to the reconstruction, there is generally no need to access the entirety of the RAW dataset. Only the events selected by the DRAW filter are needed, which represent only $\mathcal{O}(1\,\%)$ of the total dataset.

The four-layer version of the tracklet reconstruction is fully integrated in this two-phase reconstruction chain, and is actively used by ATLAS disappearing track analyses. On the other hand, the three-layer reconstruction is not a full part of this chain yet, as it was still under active development until recently. As a consequence, the three-layer reconstruction was not integrated into the `DRAW_RPVLL` filter. Simply adding it to the second phase reconstruction would not select all relevant events, since events with no four-layer but three-layer tracklets

would have been discarded by the filter beforehand. Therefore, the DRAW has to to be reproduced using the newly added three-layer reconstruction algorithm.

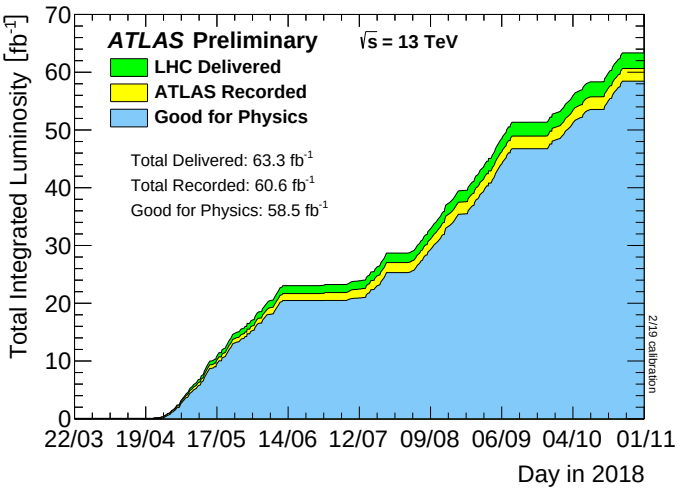


Figure 9.9.: Cumulative luminosity as a function of the day in 2018. Shown are the total luminosity delivered by the LHC, the luminosity recorded by ATLAS, as well as the fraction deemed suitable for analysis. Overall, $58.5\,\mathrm{fb}^{-1}$ of integrated luminosity fulfill this last criterion. Figure taken from [231].

Unfortunately, rerunning this DRAW filter is not an easy feat: Petabytes worth of RAW data have to be staged out of tape storage, distributed across the computing grid, and the DRAW filter program has to run on them. At the time of writing, a full reprocessing of the DRAW filter and subsequent steps was only available for the 2018 dataset. The full Run 2 dataset is currently being reprocessed, but is still pending verification. Therefore, the analysis presented in this thesis uses the 2018 dataset exclusively.

Figure 9.9 shows the integrated luminosity as a function of the day in the year 2018. In green, the cumulative luminosity that was delivered by the LHC is indicated, while yellow marks the fraction that is recorded by ATLAS. Finally, the blue graph shows the integrated luminosity that is considered suitable for further analysis. Periods of time where maintenance and other works were carried out are clearly visible as flat intervals. This final portion of the dataset amounts to $58.5\,\mathrm{fb}^{-1}$, and is used in the further parts of the analysis.

Purpose	Run number range	Trigger chain name
Signal selection	348885-350013	`HLT_xe110_pufit_xe70_L1XE50`
	350067-364292	`HLT_xe110_pufit_xe65_L1XE50`
Z^0 selection	348885-364292	`HLT_mu26_ivarmedium`
	348885-364292	`HLT_e26_lhtight_nod0_ivarloose`

Table 9.1.: Overview of triggers used for the analysis. Applicable triggers for data from years other than 2018 have been omitted.

Events used in the analysis are required to pass a specific set of trigger chains (see section 3.4.1), indicated in table 9.1. The concrete trigger chains were chosen as they are the

lowest unprescaled ones for this type of selection, and therefore yield the largest dataset, while minimizing statistical uncertainties.

For the signal selection, triggers sensitive to missing transverse energies are used. The $E_{\mathrm{T}}^{\mathrm{miss}}$ trigger names reference three different cut values, which correspond to the different algorithms that are part of the chains (see section 3.4.1 for details). The $E_{\mathrm{T}}^{\mathrm{miss}}$ threshold at L1 is 50 GeV for the entire dataset.

At the HLT, the thresholds required for the pile-up suppressed $E_{\mathrm{T}}^{\mathrm{miss}}$ value (see section 3.4.1) are 70 GeV for about the first half of 2018, while it was lowered to 65 GeV afterwards, to adjust the trigger rate. Finally, the $E_{\mathrm{T}}^{\mathrm{miss}}$ value derived from all calorimeters cells needs to be larger than 110 GeV in both cases.

Aside from the $E_{\mathrm{T}}^{\mathrm{miss}}$ triggers, two single-lepton triggers sensitive to electrons or muons are shown. These triggers select events based on leptons reconstructed at the trigger level that are above a certain p_{T} threshold. Also, particle identification and isolation criteria are applied at the HLT. Again, both of these trigger chains represent the lowest unprescaled ones for the 2018 dataset, and are used for a Z^0 selection that is needed for the scattering background estimation procedure (see section 9.7.2).

9.4.2. Simulated samples

Aside from the recorded dataset described above, MC simulation is used for certain aspects of the analysis. Data-driven methods are used almost exclusively for background estimation, meaning the use cases for MC simulation are limited.

The most notable application of MC methods is the simulation of the signal process itself. In order to obtain a signal hypothesis and predictions for observable quantities, this simulation is crucial. Various model parameters affect the signal process characteristics. The dominant one is the chargino mass, which largely drives the phenomenology. Apart from this, the mass splitting m_+ between the chargino and the two neutralinos is relevant, and is closely tied to the chargino lifetime, as was discussed in depth in section 9.2.

Selected points in the parameter space are simulated. Initial generation[2] of the mass spectrum of SUSY particles and their decay properties is performed using `ISASUGRA` [232] . Then `MadGraph5` [129] in combination with `Pythia8` [130] and `EvtGen` [131] with `A14 NNPDF23LO` [233, 234] PDFs are used for process simulation. Generated charginos do not decay in the immediate simulation, but are handed over to `Geant4` [132], to enable their decay after macroscopic propagation. Additional proton-proton collision events are included to model the pile-up conditions in data.

The relationship between Δm_+ and $\tau_{\tilde{\chi}_1^\pm}$ is given by equation (9.2) and $\tau = 1/\Gamma$. The selection of model parameters can be seen in table 9.2, where the mass of the lightest neutralino is set to a fixed value, and the chargino mass is calculated according to the mass splitting value. Since very low lifetimes τ result in a suboptimal use of generated statistics, as most simulated charginos will end up not being reconstructible, the mean chargino lifetime in its rest frame

[2]The signal sample generation uses $\mu > 0$ and $\tan\beta = 5$ parameter values.

$m_{\tilde{\chi}^0}$ [GeV]	Δm_+ [MeV]	τ [ns]
160	284	0.046
200	296	0.040
240	304	0.037

Table 9.2.: Model parameters used for the generation of MC samples used in the analysis. The mass splitting Δm_+ and lifetime τ are related via equation (9.2)

is forcibly set to a value of 0.2 ns. This enhances the statistics of reconstructible charginos. In order to recover the original lifetime distribution, a reweighting procedure is employed. Using

$$w(\tau) = \prod_{i=1}^{n} \frac{\tau_0}{\tau} \exp\left[t_i \left(\frac{1}{\tau_0} - \frac{1}{\tau}\right)\right], \tag{9.4}$$

where τ_0 is the generated lifetime and τ is the target lifetime, an arbitrary lifetime distribution can be obtained. Both are defined in the chargino rest frame. n is the number of charginos in the event and t_i is the proper decay time of the i-th chargino. During event processing, the chargino proper decay time is calculated from the decay distance in the laboratory frame and the chargino momentum. To simplify the reweighting, the equation can be rewritten like

$$w(\tau) = \exp\left[n \ln \frac{\tau_0}{\tau} + \left(\frac{1}{\tau_0} - \frac{1}{\tau}\right) \sum_{i=1}^{n} t_i\right], \tag{9.5}$$

which only requires storing two event-level quantities: the overall number of charginos and the sum of their proper decay times $\sum_{i=1}^{n} t_i$. The branching ratio of $\tilde{\chi}^{\pm} \to \pi^{\pm}\tilde{\chi}^0$ is set to 95.5 %, while the branching ratio for leptonic decays $\tilde{\chi}^{\pm} \to e^{\pm}\nu_e\tilde{\chi}^0$ ($\tilde{\chi}^{\pm} \to \mu^{\pm}\nu_\mu\tilde{\chi}^0$) is set to 3 % (1.5 %).

9.5. Object definition

In this section, the selection criteria for the physics objects used in the analysis are outlined. These objects are then considered in various ways for the signal selection, and play a role in the background estimation. A summary of the selection criteria can be found in table 9.3 for tracklets, and table 9.5 for other kinds of objects.

9.5.1. Tracklets

Tracklets are selected by a series of requirements on specific details of the reconstruction procedure and its result, in addition to more conventional kinematic and angular criteria. The general idea is to select as many tracklets as possible originating from the signal process, while rejecting background contributions caused by scattered charged objects, or random combinations.

First, tracklets are required to consist of three or more Pixel hits, that need to be located on

three or more layers of the Pixel detector. Additionally, no SCT hits can be associated to the tracklet. Holes, which are layers without a hit, where one is expected, cannot be attached to the tracklet. If a B-layer[3] hit is expected, it has to exist. This is similar to the requirement of absence of holes, but is used because, by nature of the algorithm, missing hits on the B-layer are not found by the hole-finding algorithm. Outliers cannot be associated with the tracklet. These are hits that deviate strongly from the expected intersection of the trajectory and a given sensor, and are usually indicators of a badly reconstructed track. Another requirement is that the tracklet cannot contain spoilt Pixel hits nor ganged fakes. The former are hits with a wider cluster error[4], while the latter result due to the fact that the readout channels are shared between Pixel sensors in some cases, which can result in incorrectly measured activation signals.

Requirement	Value
Transverse momentum (with VC)	$p_{\mathrm{T}} > 20\,\mathrm{GeV}$
Pseudorapidity	$0.1 < \lvert\eta\rvert < 1.9$
Transverse impact parameter	$\lvert d_0\rvert < 0.5\,\mathrm{mm}$
Longitudinal impact parameter	$\lvert z_0 \sin\theta\rvert < 0.5\,\mathrm{mm}$
Pixel layers	≥ 3
Pixel hits	≥ 3
SCT hits	0
Silicon holes	0
Outliers	0
B-layer hits (if expected)	≥ 1
Spoilt hits	0
Ganged fakes	0
Fit quality	$\chi^2_{\mathrm{prob}} > 0.5$
Isolation	$\sum_{\Delta R<0.4} p_{\mathrm{T}}^{\mathrm{track}} / p_{\mathrm{T}}^{\mathrm{tracklet}} < 0.04$
No overlap within $\Delta R < 0.4$	$N_e = 0$, $N_\mu = 0$, $N_{\text{MS-track}} = 0$, $N_j = 0$

Table 9.3.: Summary of selection criteria applied to tracklets.

Kinematically, tracklets are required to have transverse momentum in excess of 10 GeV, while their direction in the longitudinal plane needs to be inside $0.1 < |\eta| < 1.9$. The upper bound is to restrict tracklets to the central part of the Pixel detector, and be separated from the seed-level requirement of $|\eta| < 2.2$. The lower bound is to suppress muon reconstruction inefficiencies, while the upper bound removes increasing background contributions. For the transverse momentum and η, the vertex constrained values (section 8.2.1) are used.

Stringent longitudinal[5] and transverse impact parameter requirements are applied, such that $|z_0 \sin\theta| < 0.5\,\mathrm{mm}$ and $|d_0| < 0.1\,\mathrm{mm}$. Note that the transverse impact parameter requirement is on the absolute value, rather than the one divided by the associated uncertainty, as is commonly used. Due to the low number of contributing measurements, the calculated

[3] The innermost Pixel layer is conventionally called B-layer.

[4] Technically, they mark if an MS measurement has a nearby delta-day, affecting the measurement. In the ID it is reused for a broad-error flag.

[5] z_0 is commonly modified by multiplying with $\sin\theta$ before cutting to account for worse resolution in less central directions due to longer extrapolation lengths.

Cut name	Count
All	687 169 587
$p_{\mathrm{T}} > 10$ GeV	396 984 953
$0.1 < \lvert\eta\rvert < 1.9$	274 739 288
B-layer hit	189 877 691
At least 3 Pixel hits	189 204 089
At least 3 Pixel layers	188 904 931
No SCT hits	185 050 561
No silicon holes	185 050 561
No Pixel outliers	184 368 265
No Pixel spoilt hits	130 875 518
No ganged fakes	114 088 237
$\lvert d_0\rvert < 10$ mm	114 088 237
$\lvert z_0 \sin\theta\rvert < 0.5$ mm	10 169 528
Track isolated	3 979 491
No overlapping MS-track	3 975 083
No overlapping electrons	3 974 629
No overlapping muons	3 974 140
No overlapping jets	3 941 461

Table 9.4.: Number of tracklets passing the selection criteria. All tracklets shown have to be in events passing the event cleaning requirements and being selected by the trigger.

uncertainty is not expected to be a reliable and stable quantity, especially for the shorter three-layer tracklets. Both impact parameters are calculated with respect to the primary vertex, as this makes them stable against movement of the interaction region in the center of the detector. Impact parameter requirements are very efficient at removing tracklets reconstructed from random contributions. As they do not favor pointing toward the PV, their impact parameters are distributed uniformly, rather than sharply peaking at low values.

Another cut on the track fit quality in the form of $\chi^2_{\mathrm{prob}} > 0.5$ is used to suppress background tracklets, mostly arising from random combinations. Finally, tracklets are required to be isolated, meaning that event activity around them is low. This requirement is implemented by cutting on the sum of transverse momenta of tracks within a cone of $\Delta R = \sqrt{\Delta\phi^2 + \Delta\eta^2} < 0.4$ around the tracklet, that meet certain preconditions, divided by the tracklet-p_{T} itself:

$$\frac{\sum_{\Delta R<0.4} p_{\mathrm{T}}^{\mathrm{track}}}{p_{\mathrm{T}}^{\mathrm{tracklet}}} < 0.04 \tag{9.6}$$

Low values correspond to more isolated tracklets.

Finally, tracklets which overlap with any electrons, muons, MS-tracks or jets are rejected, to suppress the background from scattered particles. Here, tracklets that are near any of the aforementioned objects within $\Delta R < 0.4$ are discarded. Muons, electrons and jets have to fulfill the *baseline* requirements outlined in the following sections. MS-tracks are also required to be accepted by the criteria listed in section 9.5.3.

To accommodate the ABCD background estimation shown later on in this chapter, the selection criteria are loosened at the individual tracklet level. The $|d_0|$ threshold is increased

to $|d_0| < 10\,\text{mm}$, while the χ^2_{prob} cut is removed entirely. For tracklets entering the signal selection, the original criteria are eventually applied again, but occur after the selection of the tracklet with largest p_{T} of an event.

A summary of all of these criteria and their exact values can also be found in table 9.3. Additionally, the numbers of tracklets passing each of these selection steps are given in table 9.4. Here, the total number of tracklets is reduced from almost 7×10^8 to just under 4×10^6. The total number of tracklets entering the signal region is further reduced by the event-level criteria, described in section 9.6. The $|d_0|$ cut is seen to not result in a tracklet number reduction, which is explained by the `DRAW_RPVLL` filter already applying this criterion.

9.5.2. Electrons

Electrons and positrons are used for certain specific purposes in this analysis, such as the lepton event veto, tracklet overlap removal, as well as the electron control region (see section 9.7.1). To qualify as a *baseline* electron, it has to fulfill a loose identification threshold, which includes an additional B-layer requirement[6]. Aside from this, a value of $p_{\text{T}} > 10\,\text{GeV}$ is demanded, as is a maximum absolute pseudorapidity of $|\eta| < 2.47$ to restrict electrons to the central part of the detector, where tracking information is available. In addition, impact parameter cuts of $|d_0/\sigma_{d_0}| < 5$ and $|z_0 \sin\theta| < 0.5\,\text{mm}$ are imposed. For *signal* electrons, which are used in the background estimation of scattered leptons and the electron control region, an additional isolation requirement is applied[7].

9.5.3. Muons

Two types of reconstructed muons are considered in the analysis. As muons interact with the ID as well as the MS, reconstruction algorithms rely on information from both of these systems. In most cases, *combined* muons are chosen, which incorporate all of the available information and give the most precise measurement of the muon properties. Alternatively, muons can be reconstructed with information from the MS only. Here, quantities are measured with less precision, which makes the application of these types of muons limited.

Combined muons are required to have at least $p_{\text{T}} > 10\,\text{GeV}$, and fall within $|\eta| < 2.5$. Impact parameters requirements of $|d_0/\sigma_{d_0}| < 3$ and $|z_0 \sin\theta| < 0.5\,\text{mm}$ are applied. Additionally, medium[8] quality criteria have to be fulfilled for the muon to be considered *baseline*. Similarly to electrons, muons considered as *signal* have to pass an isolation criterion[7]. These two types of muons are used in the tracklet selection, event-level lepton veto, as well as the estimation of the scattered-lepton background and associated lepton control region.

MS-only muons are used in the tracklet selection as well, and play a central role in the tag-and-probe method used for the lepton-scatter background estimation. They are required to feature at least $p_{\text{T}} > 10\,\text{GeV}$ in addition to quality criteria to ensure well reconstructed objects.

[6] `LooseBLLHElectron`
[7] `FCLoose`
[8] `Medium`

9.5.4. Jets

Jets used in the analysis are reconstructed using the anti-k_t algorithm [125] with a radius parameter of $R = 0.4$. They are required to be central, ensured by the condition $|\eta| < 2.8$, and feature transverse momentum in excess of 25 GeV. In addition, jets have to pass the *medium* working point of the centrally implemented jet-vertex tagging algorithm, in order to suppress jets from pile-up interactions.

Cut	Electrons	Muons	Jets	MS-tracks
p_{T}	> 10 GeV	> 10 GeV	> 25 GeV	> 10 GeV
$\lvert\eta\rvert$	< 2.47	< 2.5	< 2.8	-
$\lvert d_0/\sigma_{d_0}\rvert$	< 5	< 3	-	-
$\lvert z_0 \sin\theta\rvert$	< 0.5 mm	< 0.5 mm	-	-
PID	loose + B-layer	medium	-	-
Isolation (signal)	loose	loose	-	-
other	-	-	JVT > 0.59	-

Table 9.5.: Summary of the various selection criteria that are being applied to electrons, muons, jets and MS-tracks in the analysis.

9.5.5. Overlap removal

In order to avoid any potential double-counting of physics objects defined in this section, a procedure which successively removes objects which overlap with other ones is employed. Here, the priority depends on the types of objects being compared. The procedure is implemented centrally according to a recommended prescription [235]. In summary, overlaps between electrons, jets and muons are resolved as follows:

1. Muons which share the same ID-track with an electron are removed if they are not associated with bremsstrahlung signals in the calorimeter. Otherwise, the electron is removed.

2. Jets within $\Delta R(e, j) < 0.2$ of an electron are removed and the corresponding electron is kept.

3. Electrons within $0.2 < \Delta R(e, j) < 0.4$ of jets passing (2) are discarded.

4. Muons within $\Delta R(\mu, j) < 0.2$ of a jet or *ghost-associated* with them are removed if the number of tracks of the jet $N^{j}_{\mathrm{track}}|_{p_{\mathrm{T}}>500\,\mathrm{MeV}}$ is larger than 2. Otherwise the muon is kept and the jet is discarded.

5. Muons within $0.2 < \Delta R(\mu, j) < 0.4$ of a jet passing (4) are removed.

Overlap removal between objects of the same type is typically not necessary, as the respective reconstruction routines avoid duplicates in their output already.

9.5.6. Missing transverse energy

Missing transverse energy is not strictly a physics object in an of itself. It is calculated from the objects selected and used for the analysis, as described previously in section 3.5.4:

$$\vec{E}_{\mathrm{T}}^{\mathrm{miss}} = -\sum \vec{p}_{\mathrm{T}}^{\mathrm{jet}} - \sum \vec{p}_{\mathrm{T}}^{\mathrm{electron}} - \sum \vec{p}_{\mathrm{T}}^{\mathrm{muon}} - \sum \vec{p}_{\mathrm{T}}^{\mathrm{track}} \tag{9.7}$$

ATLAS maintains a centralized implementation of the calculation algorithm, which is used for this analysis. As an input, *baseline* electrons and muons, as well as selected jets are passed into the procedure. In addition, the $E_{\mathrm{T}}^{\mathrm{miss}}$ calculation uses tracks ($\sum \vec{p}_{\mathrm{T}}^{\mathrm{track}}$) which are not associated with any of the explicitly selected objects to recover any visible E_{T}. However, tracklets are not considered in the calculation. For a signal event, where the tracklet p_{T} is closely related to the chargino p_{T}, this means that the vectorial sum of $\vec{E}_{\mathrm{T}}^{\mathrm{miss}} + \vec{p}_{\mathrm{T}}^{\mathrm{tracklet}}$ should balance in the transverse plane. As the $E_{\mathrm{T}}^{\mathrm{miss}}$ tool performs its own optimized overlap removal, it is invoked with the set of objects selected prior to the overlap removal procedure outlined in section 9.5.5.

9.6. Signal event selection

Using the objects defined in section 9.5, as well as some other quantities, an event-level selection is defined for the analysis, whose primary purpose is to obtain a dataset which contains as many events from the signal process under study as possible. At the same time, the selection should be efficient at rejecting events which arise from any of the background sources described before.

ATLAS centrally maintains a list of data-taking runs which are considered suitable for analysis, ensuring that the detector was working correctly and all necessary subsystems and components were operational. Events are required to have been recorded during such a blessed run.

Various quality criteria are applied, which reject events without a reconstructed primary vertex, and events which contain a jet that is identified as *bad*. This label groups a set of criteria to select jets originating from noise effects in the calorimeters, cosmic muon showers or beam induced background effects [236]. The presence of such jets can negatively affect other aspects of event reconstruction, such as the $E_{\mathrm{T}}^{\mathrm{miss}}$ calculation.

Events are required to have been accepted by any of the trigger chains listed in table 9.1 under *signal selection*. These triggers select events with large $E_{\mathrm{T}}^{\mathrm{miss}}$ as calculated by the trigger (*online* $E_{\mathrm{T}}^{\mathrm{miss}}$).

A requirement on the offline missing transverse energy (section 9.5.6) of $E_{\mathrm{T}}^{\mathrm{miss}} > 200\,\mathrm{GeV}$ is used. This value is chosen such that the trigger is reasonably efficient for the selected events. Due to the background estimate approach explained later in this chapter, this cut is lowered for a preliminary pass of event selection to only 100 GeV. This enables the definition of additional selection regions below the signal threshold of 200 GeV.

Another requirement is a leading-p_{T} jet, meaning the jet with the largest amount of p_{T},

with at least $p_{\mathrm{T}}^{j} > 100\,\mathrm{GeV}$. This selects events where the required ISR jet (see section 9.2) has large enough energy to provide transverse momentum to the chargino decay products via recoil, thus causing $E_{\mathrm{T}}^{\mathrm{miss}}$. To suppress events where the $E_{\mathrm{T}}^{\mathrm{miss}}$ is caused by a jet with mismeasured energy, which often arises from QCD processes, events where any selected jet is closer than $|\Delta\phi(j, E_{\mathrm{T}}^{\mathrm{miss}})| < 1.0$ to the direction of $E_{\mathrm{T}}^{\mathrm{miss}}$ in the transverse plane are rejected.

Since the signal process is not expected to contain any electrons or muons, all events with any leptons that pass the *baseline* requirements, outlined in section 9.5, are vetoed. This veto is applied in addition to the rejection of tracklets which overlap with an electron or muon.

Cut name	Count	
All	420 505 572	
Event cleaning	401 616 744	
Accepted by trigger	41 155 412	
No electrons in event	39 957 837	
No muons in event	37 232 732	
Leading jet $p_{\mathrm{T}} > 100$ GeV	28 133 095	
$E_{\mathrm{T}}^{\mathrm{miss}} > 100$ GeV	14 083 326	
$E_{\mathrm{T}}^{\mathrm{miss}}$ and leading jet separated	8 597 997	
Event has #tracklet ≥ 1	789 606	
	$N_{\mathrm{pix}} = 3$	$N_{\mathrm{pix}} \geq 4$
	782 731	6875
$E_{\mathrm{T}}^{\mathrm{miss}} > 200$ GeV	145 114	1495
$p_{\mathrm{T}} > 20$ GeV	1041	422
χ^2-probability > 0.5	406	240
$\lvert d_0 \rvert < 0.1$ mm	327	214

Table 9.6.: Number of events passing the signal event selection criteria. Events are required to have at least on tracklet. The leading-p_{T} tracklet is chosen and the final criteria are applied to this tracklet.

Finally, events have to contain at least one reconstructed tracklet passing the previously mentioned selection conditions. A summary of selection criteria at the event level can be found in table 9.7. Table 9.6 shows the number of events passing the different selection steps outlined in this section. The base selection requirements at event level reduce the overall number of events from over 4×10^8 to less than 9×10^6. By requiring at least one tracklet passing the tracklet-level selection, this number is reduced to just under 8×10^5. At this stage the leading-p_{T} tracklet, which is the tracklet with the largest p_{T}, is selected, effectively lifting it to an event-level property. After p_{T} ranking, the signal tracklet selection is completed by requiring the tighter $|d_0|$ cut, applying the cut on χ^2_{prob}, and demanding a slightly increased value of $p_{\mathrm{T}} > 20\,\mathrm{GeV}$. The nominal event-level signal region requirement of $E_{\mathrm{T}}^{\mathrm{miss}} > 200\,\mathrm{GeV}$ is applied at this point as well. Overall, this reduces the number of events entering the signal region to 327 containing a three-layer tracklet and 214 containing a four-layer tracklet.

The number of events selected as a function of the run number is shown in figure 9.10 in the top panel. Each run corresponds to a data-recording period of up to about 12 hours. The figure also indicates the integrated luminosity recorded in each run in the middle panel. The lower

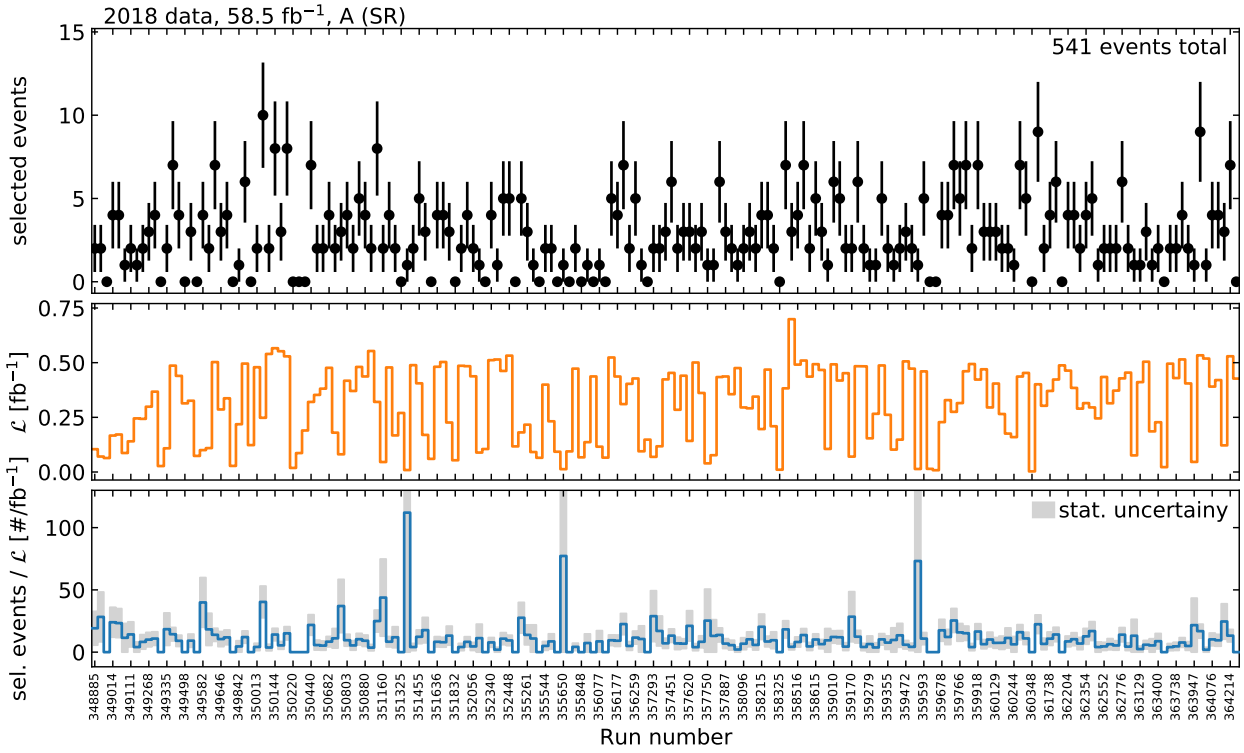


Figure 9.10.: Number of selected events as a function of run number (top panel). The middle panel shows integrated luminosity for each run. The bottom panel shows the number of selected events divided by the integrated luminosity alongside the statistical uncertainty. See figure B.1 for an expanded version.

panel shows the number of events divided by the integrated luminosity. This distribution should be approximately constant, as the probability for an event to pass the selection is independent of the time at which it is recorded. Apart from three variations, in the form of spikes, due to statistical uncertainties of the number of events per run, the ratio is reasonably stable.

Figure 9.11 displays spectra of quantities in the signal region. The figures show events containing a leading-p_{T} three- or four-layer tracklet separately. No distinction between these tracklet types is made during the selection. The missing transverse energy shown in figure 9.11a clearly shows the minimum $E_{\mathrm{T}}^{\mathrm{miss}}$ threshold of 200 GeV, and otherwise exhibits a falling spectrum. As the selection demands at least one jet in excess of $p_{\mathrm{T}} > 100$ GeV in the event, the p_{T} distribution of the jet with largest p_{T} in figure 9.11b does not show entries below that value. The distribution peaks at around 200 GeV, which corresponds to the value of the $E_{\mathrm{T}}^{\mathrm{miss}}$ requirement. As the leading-p_{T} jet acts as recoil partner for the system of particles producing the missing transverse energy, they are expected to be approximately balanced. This is reinforced by the $E_{\mathrm{T}}^{\mathrm{miss}}$-jet separation requirement. It is visualized by the $\Delta\phi(E_{\mathrm{T}}^{\mathrm{miss}}, j_1)$ in figure 9.11c distribution, which corresponds to the angle in the transverse plane between $E_{\mathrm{T}}^{\mathrm{miss}}$ and the leading-p_{T} jet. Events with $|\Delta\phi(E_{\mathrm{T}}^{\mathrm{miss}}, j_1)| < 1$ are rejected. The distribution rises toward $|\Delta\phi(E_{\mathrm{T}}^{\mathrm{miss}}, j_1)| \sim \pi$, corresponding to the jet and $E_{\mathrm{T}}^{\mathrm{miss}}$ pointing in opposite

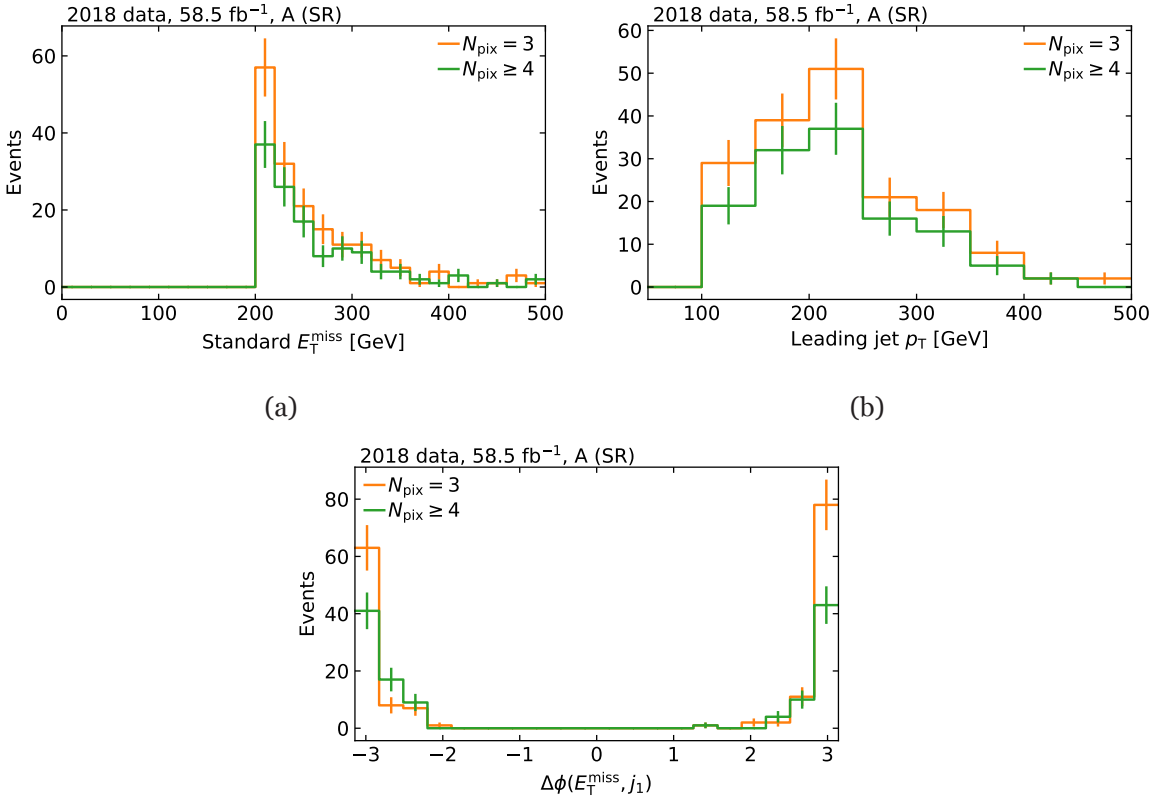


Figure 9.11.: Distributions of the default E_T^{miss}, the p_T of the leading-p_T jet and the separation between the two objects, for events passing the signal selection. Events with three- and four-layer tracklets are drawn separately.

directions in the transverse plane.

9.7. Template based background estimation

The background estimation method described in this section uses a set of histograms describing the shapes of the spectra of tracklet p_T in the signal region. For each background source, a dedicated histogram, called *template*, is derived. Different strategies are used for the their production depending on the process. This section introduces the relevant template derivation procedures, as well as a strategy to combine them into a complete background estimate.

9.7.1. Lepton and hadron control regions

The template derivations outlined in this section rely on the calculation of a set of transfer factors that convert a sample of well-reconstructed objects to an estimate of a scattered object sample. To this end, dedicated control regions are defined to obtain the needed sample of objects to apply the transfer factors to. Their basic structure resembles the common signal

event selection. Instead of a tracklet, an identified electron or muon fulfilling the *signal* selection criteria defined in sections 9.5.2 and 9.5.3 is required. For the hadron control region, standard tracks are selected instead of a tracklet. The track is required to pass the same kinematic and quality selection criteria, with the exception of the condition of no SCT hits. As the prescaled lepton filter is not desirable in this case, the lepton and hadron control regions do not use the dataset reconstructed from the `DRAW_RPVLL` output. No tracklets are required for the control regions, therefore a dataset from the standard RAW is chosen.

It was not studied whether applying the track level criteria also to the selected leptons improves the control region selection. Under the assumption that the effect of these selections are captured in the transfer factor determination described below, it is expected to have a small impact. Nevertheless, refinement of the lepton selection could be a future improvement to the analysis.

Modified missing transverse energy

In the regular signal selection, the missing transverse energy of the event is required to exceed a value of 200 GeV. Since the tracklet is not included in the calculation of the $E_{\mathrm{T}}^{\mathrm{miss}}$, its transverse momentum manifests itself as an invisible contribution. The same is not the case for the control regions, as leptons and hadrons enter the $E_{\mathrm{T}}^{\mathrm{miss}}$ calculation as visible. To make the selected sample of events more similar to the signal region kinematically, the default $E_{\mathrm{T}}^{\mathrm{miss}}$ calculation is modified. For the lepton control regions, the *signal* lepton is explicitly treated as invisible in the conventional calculation, yielding $E_{\mathrm{T}}^{\mathrm{miss}}|_{\not{\mathrm{lep}}}$. In the hadron control region, the $E_{\mathrm{T}}^{\mathrm{miss}}$ calculation is conducted nominally, and the transverse momentum of the track is added to the result like

$$\vec{E}_{\mathrm{T}}^{\mathrm{miss}}|_{\not{\mathrm{trk}}} = \vec{E}_{\mathrm{T}}^{\mathrm{miss}} + \vec{p}_{\mathrm{T}}^{\,\mathrm{track}}. \tag{9.8}$$

The same threshold requirement of $E_{\mathrm{T}}^{\mathrm{miss}} > 200$ GeV is applied on these modified $E_{\mathrm{T}}^{\mathrm{miss}}|_{\not{\mathrm{lep}}}$ and $E_{\mathrm{T}}^{\mathrm{miss}}|_{\not{\mathrm{trk}}}$ values. Distributions of the standard $E_{\mathrm{T}}^{\mathrm{miss}}$ and the modified values for electron, muon and hadron control regions are shown in figure 9.12. It can be observed that the standard $E_{\mathrm{T}}^{\mathrm{miss}}$ tends to have smaller values as compared to the modified values. The modified $E_{\mathrm{T}}^{\mathrm{miss}}$ start at 100 GeV, which is due to the fact that this is the basic requirement placed on events. It differs from the true signal region threshold of 200 GeV which is applied at a later stage.

Assuming an event topology with a recoil jet, leptons or tracks are likely to point in the general direction as any real $E_{\mathrm{T}}^{\mathrm{miss}}$ found in the event. By excluding the *signal* lepton or track, the maximum is shifted to larger values, and increases the fraction of events featuring $E_{\mathrm{T}}^{\mathrm{miss}}$ values above 200 GeV.

Table 9.7 shows an overview of how these different $E_{\mathrm{T}}^{\mathrm{miss}}$ values are used in the event-level selections for the signal region and the lepton and hadron control regions. Other differences between the selections, like the different configurations of the lepton vetoes, are indicated as well.

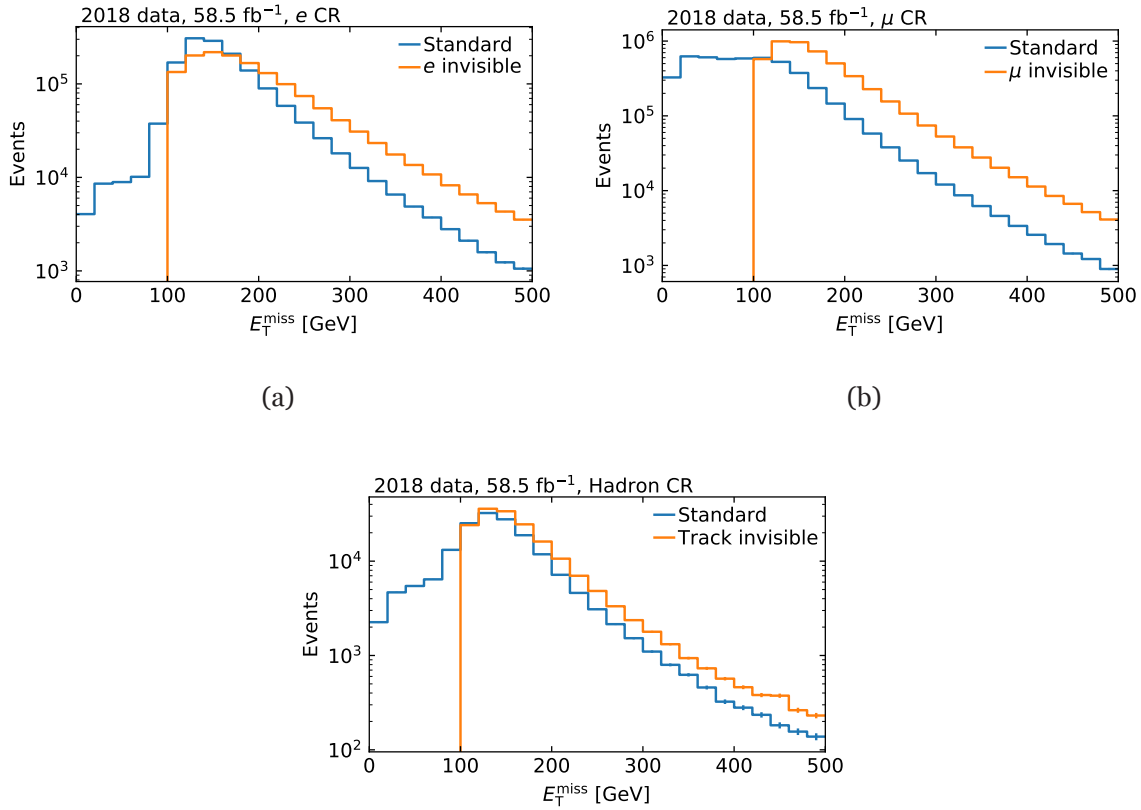


Figure 9.12.: Distributions of different values of $E_{\mathrm{T}}^{\mathrm{miss}}$ for the electron, muon and hadron control regions. Shown is the $E_{\mathrm{T}}^{\mathrm{miss}}$ value obtained where all leptons and tracks enter as visible, as well as the value where the *signal* lepton or track is treated as invisible ($E_{\mathrm{T}}^{\mathrm{miss}}|_{\not{\mathrm{lep}}}$, $E_{\mathrm{T}}^{\mathrm{miss}}|_{\not{\mathrm{trk}}}$).

Requirement	Tracklet	Electron	Muon	Hadron
Trigger	Accepted by $E_{\mathrm{T}}^{\mathrm{miss}}$ chain			
$E_{\mathrm{T}}^{\mathrm{miss}}\|_X > 200\,\mathrm{GeV}$	$E_{\mathrm{T}}^{\mathrm{miss}}$	$E_{\mathrm{T}}^{\mathrm{miss}}\|_{\not{e}}$	$E_{\mathrm{T}}^{\mathrm{miss}}\|_{\not{\mu}}$	$E_{\mathrm{T}}^{\mathrm{miss}}\|_{\not{\mathrm{trk}}}$
$\max(p_{\mathrm{T}}^{j})$	$> 100\,\mathrm{GeV}$			
$\min(\Delta\phi(j, X)) > 1$	$E_{\mathrm{T}}^{\mathrm{miss}}$	$E_{\mathrm{T}}^{\mathrm{miss}}\|_{\not{e}}$	$E_{\mathrm{T}}^{\mathrm{miss}}\|_{\not{\mu}}$	$E_{\mathrm{T}}^{\mathrm{miss}}\|_{\not{\mathrm{trk}}}$
Leptons	$N_e = 0$ $N_\mu = 0$	$N_e = 1$ $N_\mu = 0$	$N_e = 0$ $N_\mu = 1$	$N_e = 0$ $N_\mu = 0$

Table 9.7.: Overview of event-level requirements being applied in the signal tracklet selection, as well as the lepton control regions.

Lepton transverse momentum smearing

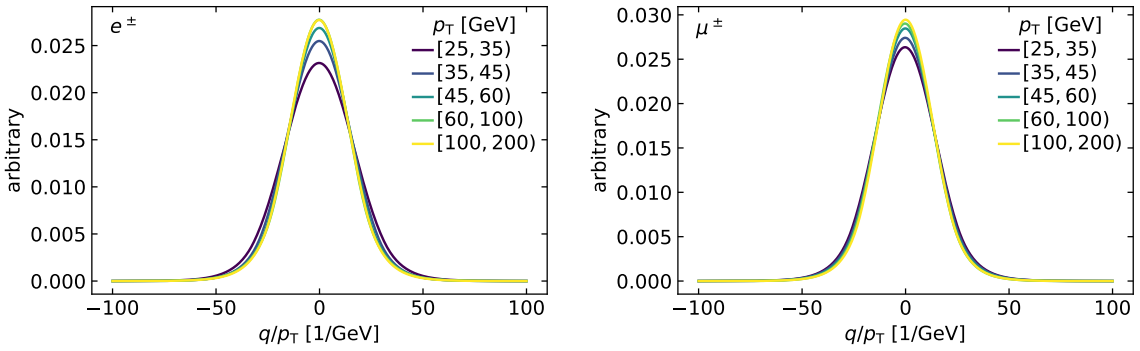
One major discrepancy between standard tracks and signal tracklets needs to be considered. Since standard tracks consist of a larger number of measurements, and are generally longer, their p_{T} resolution is considerably better than that of tracklets. This was discussed in detail in section 8.2.1. However, even when using the value obtained from the vertex constrained refit,

the discrepancy remains. To be able to convert from a standard-track-based p_{T} spectrum to one compatible with the tracklet p_{T} spectrum, a smearing procedure as described in [219] is applied. As both electrons and muons are reconstructed with standard tracks as an ingredient, in additional to flavor-specific signatures from the calorimeter and the MS, the same procedure is used.

For this smearing, the relative resolution between tracklets and standard tracks is required. The resolutions are obtained by using standard tracks, and refitting them considering only their associated Pixel hits in $Z^0 \to l^+l^-$ events. By comparing the q/p_{T} values of the original track and the refitted tracklet, a resolution distribution can be derived. A double-sided crystal-ball function in the form of

$$f(x,z,\sigma,\alpha) = \left[\sigma\sqrt{2\pi}\,\mathrm{erf}\left(\frac{\alpha}{\sqrt{2}}\right) + \frac{2}{\alpha}e^{\frac{-\alpha^2}{2}}\right]^{-1} \cdot \begin{cases} e^{\frac{1}{2}\left(\frac{x-z}{\sigma}\right)^2} & -\alpha \leq \frac{x-z}{\sigma} \leq \alpha \\ e^{\alpha\left(\frac{x-z}{\sigma}+\frac{\alpha}{2}\right)} & \frac{x-z}{\sigma} < -\alpha \\ e^{-\alpha\left(\frac{x-z}{\sigma}-\frac{\alpha}{2}\right)} & \frac{x-z}{\sigma} > \alpha \end{cases} \tag{9.9}$$

is fitted to the distribution. It is chosen to model the central gaussian shape of the distribution, as well as the slopes on both sides. Fits for five ranges of the original track p_{T} are performed centrally, and their results are shown in figure 9.13, while the exact parameters can be found in table B.2.



(a) $e^{\pm}$ (b) $\mu^{\pm}$

Figure 9.13.: Probability density function used for smearing tracklet p_{T} values, following a double-sided crystal-ball function from equation (9.9). Values for the p_{T} ranges can be found in table B.2. (a) shows the function used for electrons, while (b) is used for muons.

The extracted functions can be used to calculate the smeared p_{T} spectrum like

$$p_{\mathrm{T}}^{\mathrm{smeared}} = \left[\frac{q}{p_{\mathrm{T}}^{l}} + R\left(\frac{q}{p_{\mathrm{T}}^{l}}\right)\right]^{-1}, \tag{9.10}$$

where q/p_{T}^{l} is the original transverse momentum related parameter of the lepton, and $R(q/p_{\mathrm{T}}^{l})$ is a random number sampled from the curves in figure 9.13.

This implementation of the analysis uses already calculated fit parameters for the resolution function, which were obtained in the way described above [237]. By refitting the track and simply discarding the SCT hits, the momentum resolution is only consistent with four-layer tracklets. However, lacking a dedicated routine to perform a refit with only three Pixel hits, this smearing function is used regardless. Figure 9.14 shows distributions of the original p_{T} of *signal* electrons and muons in the lepton control regions for these flavors, in comparison to their respective smeared values. It can be observed that the unsmeared distribution for muons shows a much weaker drop until about 125 GeV than is the case for electrons.

In previous analyses [219], the hadron and electron templates were found to be very similar after application of transfer factors and momentum smearing. In order to validate the background estimation approach and to reduce complexity, this analysis uses the electron transfer factors and momentum smearing functions to derive a hadron template. They are applied to the hadron control region track to yield the final shape histogram. Looking at figure 9.14, the shape of smeared p_{T} distributions shows some discrepancy between electrons and hadrons, although the overall trend is somewhat comparable. Nevertheless, this hadron template is used in subsequent steps. A derivation of a dedicated hadron template and corresponding transfer factors and momentum smearing functions remains as a future improvement of the analysis.

In all cases, the bulk of the distribution is shifted toward lower values. In the case of the electron spectra, at the high end of the spectrum, the population is increased for smeared electrons. For muons, the bulk of the population at medium to high values is shifted to the left. The transverse momentum cut at 10 GeV can be observed. However, due to the smearing, some migration occurs toward lower values. For the final *signal* lepton and track selections, an additional cut on the smeared p_{T} values of $p_{\mathrm{T}} > 20$ GeV is employed, rendering this effect irrelevant.

9.7.2. Scattered lepton template

As introduced in section 9.3.1, one source of background are objects such as charged leptons[9]. To estimate the distribution of events, where such a lepton scatters to produce a signature consistent with a disappearing track, a *tag-and-probe* method is used. Effectively, this methods calculates the probability for a lepton to cause a reconstructed tracklet, as a function of lepton properties.

Tag-and-probe selection

A dataset containing predominantly events with $Z^0 \to l^+l^-$ decays is used for the determination of transfer factors. Events are required to have been accepted by a set of single-lepton triggers (see table 9.1). One of the leptons from the Z^0 decay is used as the *tag*. To qualify as a tag lepton, it needs to fulfill the *signal* requirements defined in sections 9.5.2 and 9.5.3. Additionally, the tag candidate needs to have $p_{\mathrm{T}} > 30$ GeV, and be the object identified by

[9] Electrons and muons only, as taus are not treated explicitly. See section 9.7.3.

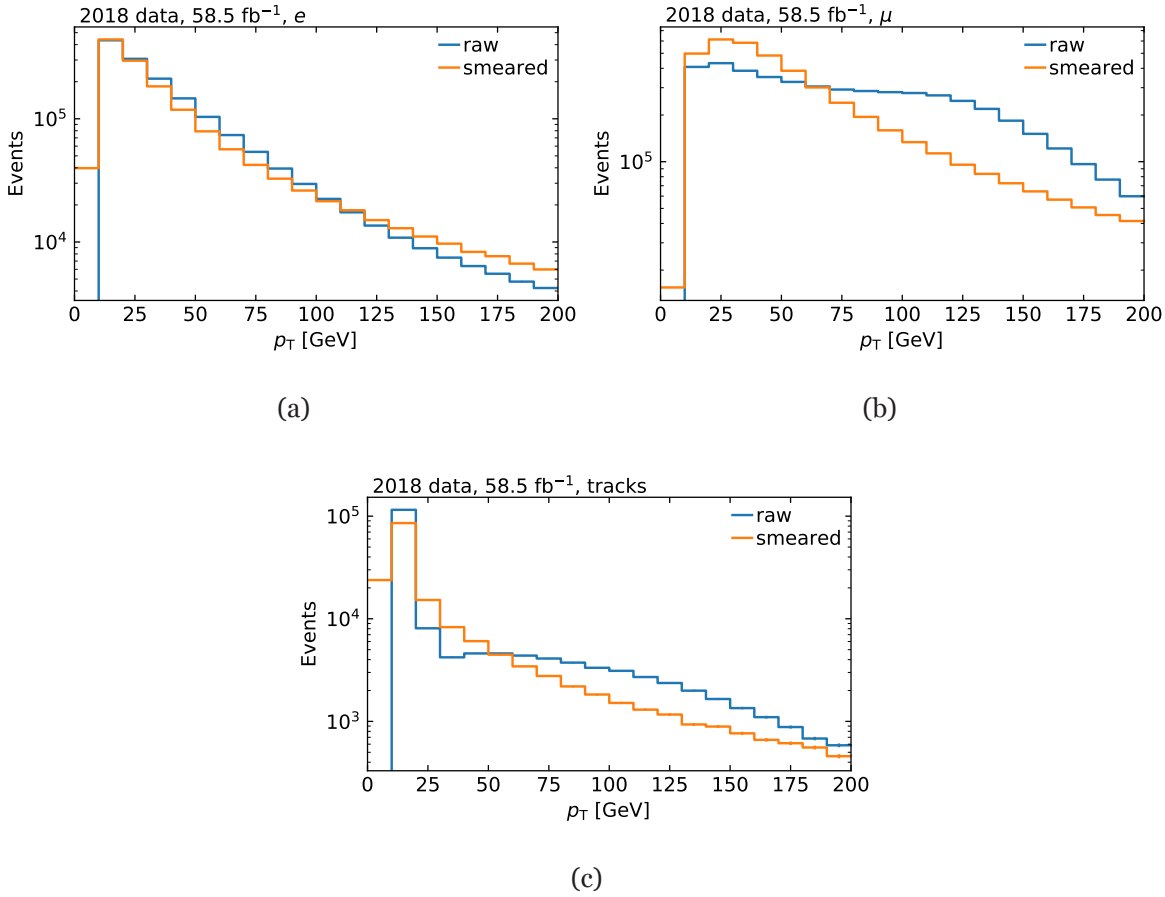


Figure 9.14.: Distributions of the original (unsmeared) p_T values of *signal* electrons, muons and standard tracks. Also shown is the result of the smearing applied to the nominal distributions, which modifies the spectra to be more comparable to the tracklet momentum resolution.

the single-lepton trigger chain for the given lepton type. This trigger matching uses the cone distance $\Delta R = \sqrt{\Delta\eta^2 + \Delta\phi^2}$ to associate offline particles to particles at trigger level. An offline particle is considered matched if it is associated with the trigger-level particle identified by the demanded trigger chain.

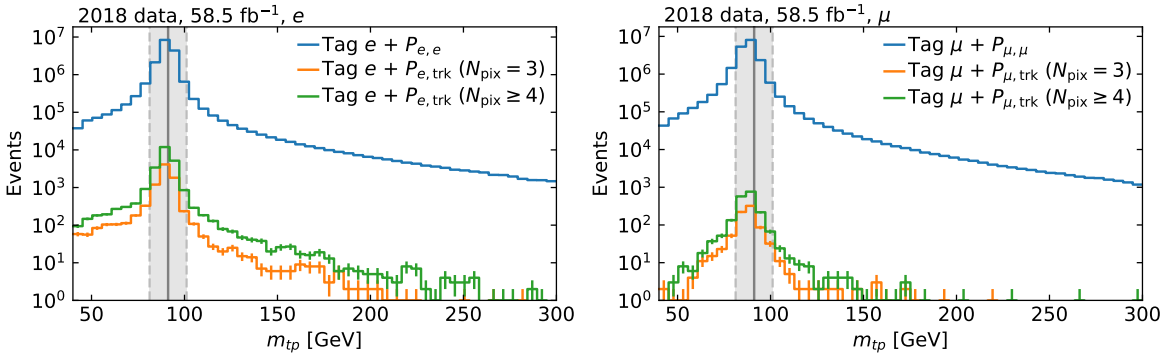
Independently of this, a set of probes is selected, whose definition intentionally aims to be as loose as possible. For electrons, raw calorimeter clusters are used, while MS-tracks serve as probes for the muon case. Both types are required to pass $p_T > 10$ GeV and $|\eta| < 2.5$. Furthermore, probes are then selected for compatibility with the Z^0 decay, by pairing them with the identified tag lepton. As a first step, tag-probe pairs are considered if they are separated in transverse plane like $|\Delta\phi(l_t, l_p)| > 1.5$. This requirement eliminates pairs where the tag and the probe actually stem from the same physical lepton. Since the probe criteria are a subset of the tag criteria (as electrons and combined muons also require the presence of a calorimeter cluster and MS-track, respectively), this is possible and expected. Subsequently, the algorithm searches for leptons, passing the *signal* selection, and tracklets in a cone of $\Delta R = \sqrt{\Delta\eta^2 + \Delta\phi^2} < 0.2$ around each probe. For tracklets, the four-momentum as obtained

using the vertex constraint (section 8.2.1) is used for the overlap calculation. According to these overlapping objects, the probes can be classified further. An overview of the different classes of probes used in the transfer calculation is given in table 9.8.

Label	Tag object	Probe object	Object overlapping with probe
$P_{e,e}$	Electron	Calorimeter cluster	Electron
$P_{e,\text{trk}}$	Electron	Calorimeter cluster	Tracklet
$P_{\mu,\mu}$	Muon	MS-track	Muon
$P_{\mu,\text{trk}}$	Muon	MS-track	Tracklet

Table 9.8.: Overview of the classification of probe objects based on overlapping objects found within $\Delta R < 0.2$

If a tag-probe pair has a combined invariant mass fulfilling $|m_{\text{tp}} - m_Z| < 10\,\text{GeV}$, where $m_Z \simeq 91.19\,\text{GeV}$ is the Z^0 mass, it is accepted for further processing. This requirement ensures a reasonably large likelihood of the probe being associated to a physical lepton from the Z^0 decay, without explicitly requiring that it has been fully reconstructed as such. Figure 9.15 shows plots of the distributions of the invariant mass of the tag-probe system m_{tp}. A clear peak of the distributions around m_Z can be observed, with the typical falling slope toward smaller and larger values, corresponding to off-shell Z^0 production, among others. Additionally, the mass window of width $\pm 10\,\text{GeV}$ centered around m_Z is drawn. Pairs inside this area are the ones that are accepted.

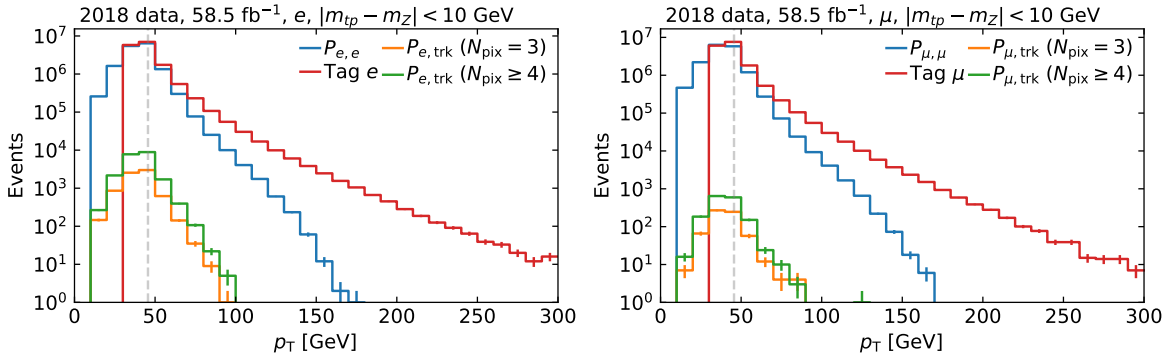


(a) e^+e^- (b) $\mu^+\mu^-$

Figure 9.15.: Invariant mass distributions of tag-probe pairs. The different probe types are overlaid. Vertical lines indicate the mass window that is used to further select tag-probe pairs

Distributions of p_{T} and η of tags and probes for both lepton flavors are shown in figures 9.16 and 9.17. Here, the different probe types, either overlapping with leptons or with tracklets, are displayed separately (see table 9.8). A probe object, such as a calorimeter cluster or MS-track, can therefore be shown multiple times if it overlaps both with a lepton and with a tracklet. All probe types show a clear peak around 45 GeV, corresponding to half of the on-shell Z^0 mass being carried by the decay products in the transverse plane. The overall number of probes overlapping with identified leptons is larger than the one for probes overlapping with

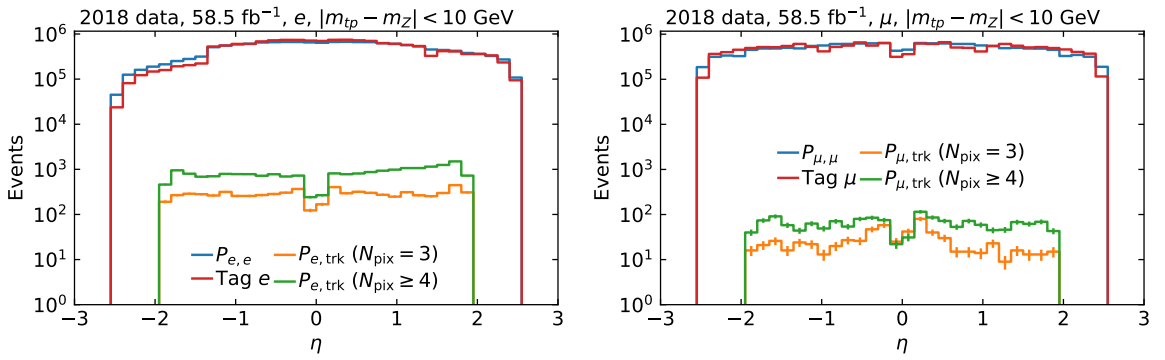
tracklets. This discrepancy is expected, as the scattering rate should be relatively low. Both minimum p_T cuts for tags and probes can be seen in figure 9.16.



(a) $e^{\pm}$ (b) $\mu^{\pm}$

Figure 9.16.: Distributions of tag and probe p_T. Both have to fulfill the individual selection criteria in addition to the mass window cut $|m_{tp} - m_Z| < 10$ GeV for each lepton flavor. The different probe types are overlaid.

The η plots in figure 9.17 show broad and flat distributions for both tags and probes. In the electron case (figure 9.17a), an asymmetry can be seen on the negative side of the spectrum. This decrease is caused by an error in the filtering mentioned in section 9.4.1. In this procedure, only events with preselected leptons are accepted. Here, one criterion on the η values of reconstructed electrons was applied incorrectly. As the transfer factor calculation output is a ratio of quantities measured in this dataset, the results should be insensitive to this regression. Aside from this, the lepton control regions (see section 9.7.1) use a dataset that is not filtered in this way, and is therefore unaffected. Generally, tags and probes with an overlapping tracklet present very similar distributions. They are found at lower rates, as is expected and cut off at $|\eta| = 1.9$, corresponding to the requirement of the tracklet selection.



(a) $e^{\pm}$ (b) $\mu^{\pm}$

Figure 9.17.: Distributions of tag and probe η. Both have to fulfill the individual selection criteria in addition to the mass window cut $|m_{tp} - m_Z| < 10$ GeV for each lepton flavor. The different probe types are overlaid.

Transfer factor calculation

To determinate a template for the background arising from scattered leptons, the probability for a lepton to scatter and be recovered as a tracklets is required. Using the selections described before, the fraction of probes overlapping with tracklets over probes overlapping with leptons,

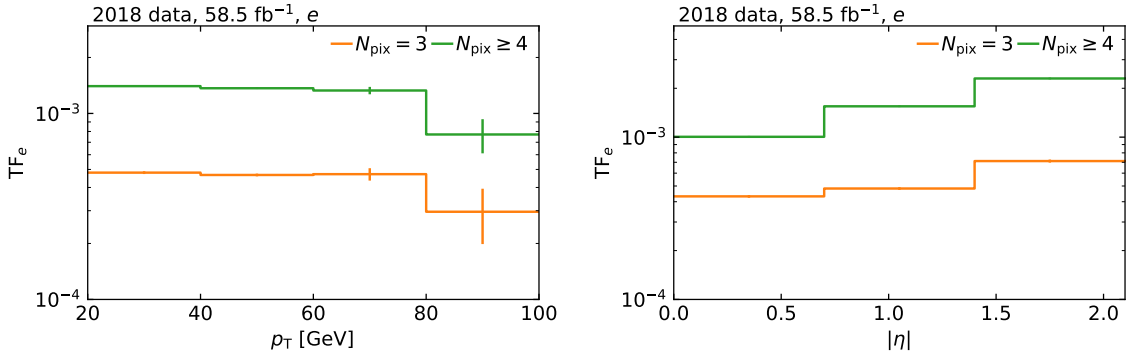
$$\mathrm{TF}_l = \frac{N(P_{l,\mathrm{trk}})}{N(P_{l,l})}, \tag{9.11}$$

can be calculated. By normalizing the number of overlapping tracklets by the number of overlapping leptons, this transfer factor can be applied to a selection of the latter object type. A sample of leptons can be converted into an estimate of the corresponding scattered lepton sample using this approach. The resulting distribution resembles that of tracklets reconstructed from scattered charged leptons. As the transfer factor is found to be dependent on the transverse momentum and longitudinal direction of the probe, the calculation is done in intervals of these quantities. Consequently, a $p_\mathrm{T} - \eta$ dependent lookup is performed later on. In addition, the transfer factor calculation is performed separately for three- and four-layer tracklets, as they are expected to differ. The number of contributing hits dictates the minimum and maximum travel distance of a lepton scatter candidate, as well as the material it traverses. In the three-layer case, the scatter event needs to happen between the third and fourth Pixel layer, as it would result in a four-layer tracklet otherwise. In case of four-layer tracklets, the entire volume between the Pixel and SCT is available for decay.

Figure 9.18 show the resulting transfer factors for electrons, while figure 9.19 shows the corresponding muon values. The uncertainties are dominated by the low statistics of probes with associated tracklets. As the population of probes overlapping with tracklets is very low above about 100 GeV, the highest bin shown is used for momentum values above this threshold as well. The corresponding statistical uncertainty in this bin is still large, especially in the muon case.

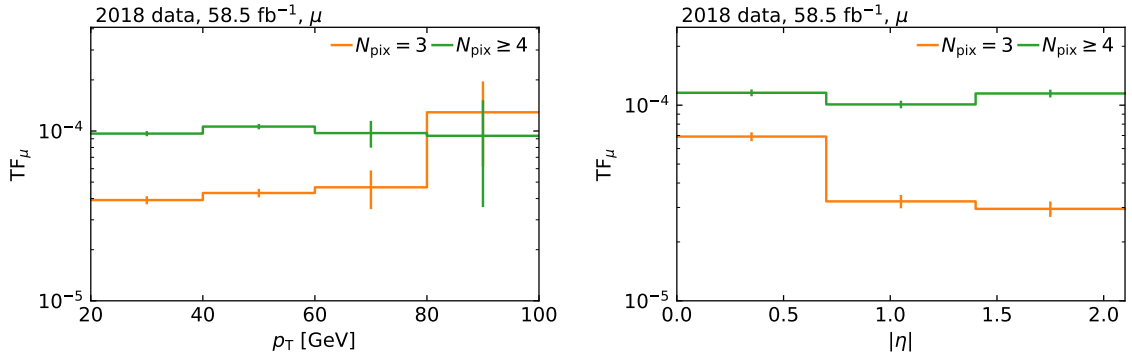
For electrons, the transfer factors decrease slightly as a function of p_T, while rising toward larger values of $|\eta|$. Overall, the transfer factors are $\mathcal{O}(10^{-3})$, while the values for three-layer tracklets are lower than the ones for four-layer tracklets, as expected. For muons, the transfer factors are on the order of $\mathcal{O}(10^{-4})$. The values are approximately constant as a function of p_T for three layers, while rising slightly for four layers. In $|\eta|$, the values for three-layer tracklets drop toward larger values, while the four-layer transfer factors remain approximately constant.

In case of the muon transfer factor, another aspect has to be taken into account. A reconstructed combined muon is composed of an ID-track and an MS-track. In the transfer factor calculation, MS-tracks are used as probe objects in the tag-and-probe method. In the tracklet selection, however, tracklets are explicitly required not to overlap with MS-tracks, in order to reduce the scatter background. Clearly, this requirement has to be ignored when performing the tag-and-probe procedure. Otherwise the denominator in equation (9.11) would vanish. This means that an additional factor accounting for the probability that a muon is reconstructed, but no MS-track is found, has to be added to the transfer factor. This additional



(a) (b)

Figure 9.18.: Transfer factors as a function of calorimeter cluster p_{T} and η for electrons. Three- and four-layer tracklet transfer factors are shown separately.



(a) (b)

Figure 9.19.: Transfer factors as a function of MS-track p_{T} and η for muons. Three- and four-layer tracklet transfer factors are shown separately.

factor is taken from the result of a derivation performed outside of the work of this thesis [237]. It is found to be small, as can be seen in figure B.3, reflecting the very low probability for a muon to undergo strong scattering, and not producing a track in the MS. Overall this results in a very small expected number of muon scatter events.

As a final step, the measurement of the probability that a lepton scatters needs to be translated into an actual estimation of the event yield contributing to the signal region. Such an estimate can be obtained using the dedicated lepton Control Regions (CRs) defined in section 9.7.1. For each *signal* lepton selected in these control regions, a lookup is performed using a two-dimensional p_{T}-$|\eta|$ histogram (see figure B.2) of the transfer factors discussed just above. This factor is then used as a weight when populating histograms with the *signal* lepton. Note that the lookup is performed using the p_{T} and $|\eta|$ values of the underlying probe object, calorimeter clusters for electrons and MS-tracks for muons. This is to ensure a consistent lookup, as the probe object properties are used in the calculation of the transfer factors.

9.7.3. Scattered hadron template

As is the case for leptons, charged hadrons travelling through the detector have a probability to undergo scattering. Such a scatter event can be severe enough to affect the reconstruction of the hadron. A major source of hadrons in the context of this analysis is via the production and subsequent hadronic decay of τ-leptons. In a large fraction of cases, the result is a number of charged pions, which are detected by the tracking system. Additionally, in case of τ production via $W^{\pm}$ bosons, the final state also contains a neutrino which enter the missing transverse energy of the event. No dedicated pion reconstruction and identification algorithms are commonly used in ATLAS. To attempt a quantification of this hadron background, standard tracks are selected. By requiring no leptons or jets in the vicinity of the tracks, sources other than charged pions can be suppressed.

For a complete description of the hadron background using a transfer factor based approach, a dedicated calculation is necessary. In order to validate the background estimation procedure outlined in this section, the true hadron transfer factor is approximated by the electron transfer factor. This is motivated by the conclusion that hadron and electron shapes are reasonably similar made by previous analyses [219]. The hadron control region sample described in section 9.7.1 is used to obtain a background template by the application of the electron transfer factor.

9.7.4. Combinatorial fake template

Random combinations of measurements can be reconstructed as tracklets. The probability for this is larger for three-layer tracklets, as there are more random triplet combinations than there are quadruplet combinations for a given number of measurements. Such fake tracklets that pass the signal selection constitute a background.

The derivation of a fake background template is based on the observation that random combinations do not favor pointing toward the interaction point. Aside from a very loose requirement of general compatibility, no assumptions on the impact parameters relative to the primary vertex are made at the reconstruction level. The impact parameter distributions of fake tracklets are therefore expected to be mostly uniform, both in d_0 as well as z_0. This can be seen in figures 8.10 and 8.11 in section 8.2.2. In the signal MC sample shown there, tracklets not originating from the signal process are assumed to be largely dominated by random combinations. At the analysis selection level, low absolute values of both impact parameters are required.

Another parameter of interest is the χ^2-probability, which is a property related to the goodness of the trajectory fit to the set of measurements. Figure 8.12 shows that signal tracklets tend toward lower values, whereas other tracklets are mostly found at larger values. An analysis-level cut of $\chi^2_{\text{prob}} > 0.5$ is used.

To derive a template for the fake background, a *tracklet-quality* sideband is defined, by inverting either one or both of the aforementioned criteria. In all cases, this makes the sideband selection orthogonal to the signal selection. The distribution obtained in this sideband

is used as a template for the fake background.

9.7.5. Background normalization

Following the template based approach, the strategy is to determine their absolute normalization in a fit to the data. A likelihood based fit method is used, that is explained in more detail in subsequent sections. To this end, another control region is defined, which differs from the signal region only by a missing transverse energy range of 120 GeV $< E_{\mathrm{T}}^{\mathrm{miss}} <$ 180 GeV rather than the cut of $>$ 200 GeV. An additional validation region is foreseen at an intermediate range of 180 GeV $< E_{\mathrm{T}}^{\mathrm{miss}} <$ 200 GeV.

The region layout is shown in figure 9.20, which also indicates the corresponding tracklet quality sidebands used to determine the fake background templates. As the only difference in the ”pass” criteria shown in the upper row is $E_{\mathrm{T}}^{\mathrm{miss}}$, the template derivation procedures described above can be used to construct individual templates for all regions. For the fake templates, the respective $E_{\mathrm{T}}^{\mathrm{miss}}$ cuts are kept, and the tracklet uality ”fail” criteria are applied.

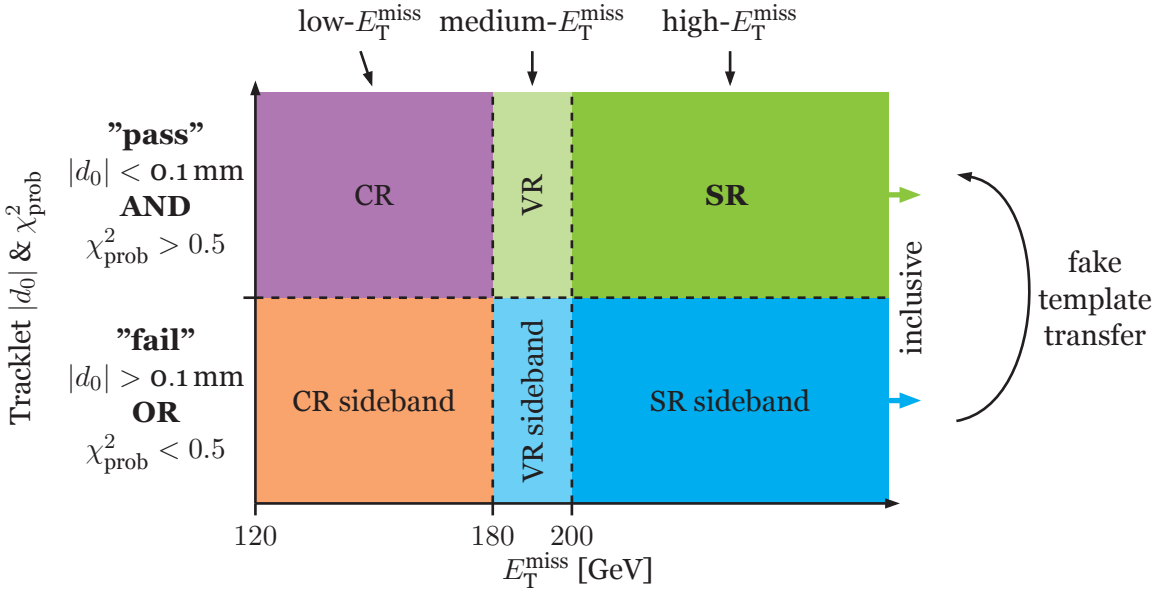
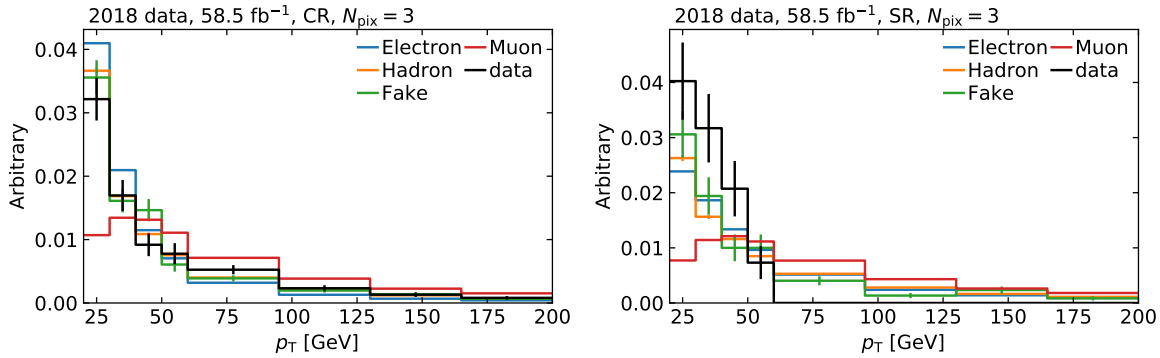


Figure 9.20.: Schematic of the *tracklet-quality* sidebands used to obtain templates for the fake background. A low-$E_{\mathrm{T}}^{\mathrm{miss}}$ control region, a medium-$E_{\mathrm{T}}^{\mathrm{miss}}$ validation region and a high-$E_{\mathrm{T}}^{\mathrm{miss}}$ signal region are defined. The sideband is given by inversion of either one or both of the d_0 or χ^2_{prob} criteria.

The fit procedure is set up to vary the normalization of each template individually, with an unconstrained factor that affects both the control and signal region in a correlated way. Three- and four-layer tracklet histograms are fit independently. While the fit procedure does converge overall, it was found that the templates do not feature sufficient shape differences to allow the fit to determine their relative contributions.

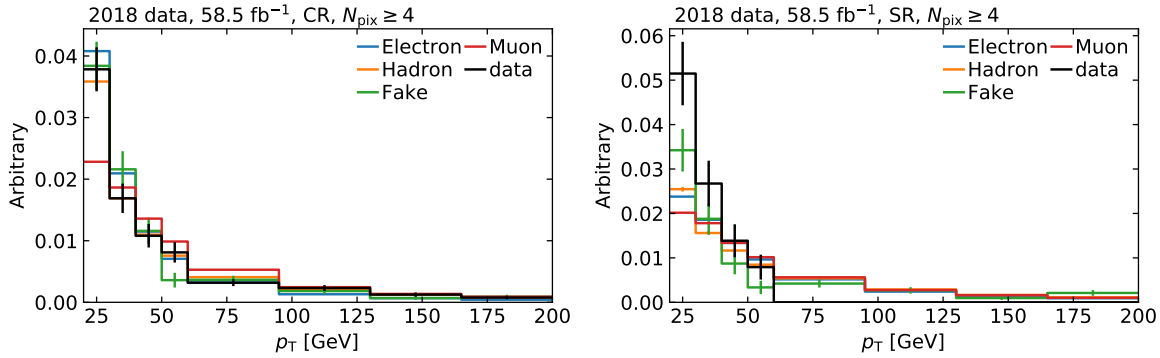
Figures 9.21 and 9.22 show the template shapes for three- and four-layer tracklets, in the form of normalized p_{T} distributions. Both the CR and the Signal Region (SR) are shown. In

addition, the selected data in both regions is shown in black. Note that in the SR, the data is blinded for $p_T > 60$ GeV. For three-layer tracklets, the templates rise toward low momenta, with the exception of the muon template, which peaks at about $p_T = 30\,\text{GeV} - 50\,\text{GeV}$. In the four-layer case, the muon template also rises toward lower values.



(a) (b)

Figure 9.21.: Normalized p_T distributions in the control and signal regions for three-layer tracklets. Shown are the data yields, as well as the different background templates. Note that the data in the signal region is blinded above 60 GeV.



(a) (b)

Figure 9.22.: Normalized p_T distributions in the control and signal regions for four-layer tracklets. Shown are the data yields, as well as the different background templates. Note that the data in the signal region is blinded above 60 GeV.

It can be observed that the shape variation between the various templates is low. As a consequence, the fit has to arbitrarily decide which background sources to prefer and which ones to suppress. This leads to an unstable background composition result, which decreases confidence in the background estimation methods. Various mitigation strategies were evaluated, ultimately resulting in the abandonment of this fit normalization approach in favor of a broader one described in the following section 9.8.

9.8. Unified total background estimation

In order to mitigate issues encountered in the determination of the normalization of different background contributions, a unified background estimation technique was developed. Rather than attempting to determine individual contributions of different sources of backgrounds, this approach estimates the total amount of background.

In a procedure called ABCD, the observed number of events in three additional regions is used to predict the number of background events in the signal region. It is crucial that these additional regions are both essentially free of events from the signal process, and orthogonal to the signal region. Conventionally, in ABCD, the signal region is assigned the A label, while the auxiliary regions are referred to as B, C and D.

The region definitions are slightly modified compared to the template based background estimation approach (see figure 9.20). The $E_{\mathrm{T}}^{\mathrm{miss}}$ criteria are the same, as are the ones related to tracklet-quality. A modified drawing showing the regions can be found in figure 9.23, with the naming following the ABCD convention. Furthermore, the A region (high-$E_{\mathrm{T}}^{\mathrm{miss}}$, passing tracklet-quality), is split into a low-p_{T} region A_L and a high-p_{T} region A_H, with the boundary at $p_{\mathrm{T}} = 60\,\mathrm{GeV}$. Signal contamination in A_L is expected to be minimal, as estimated from MC, while A_H is expected to contain the majority of signal events.

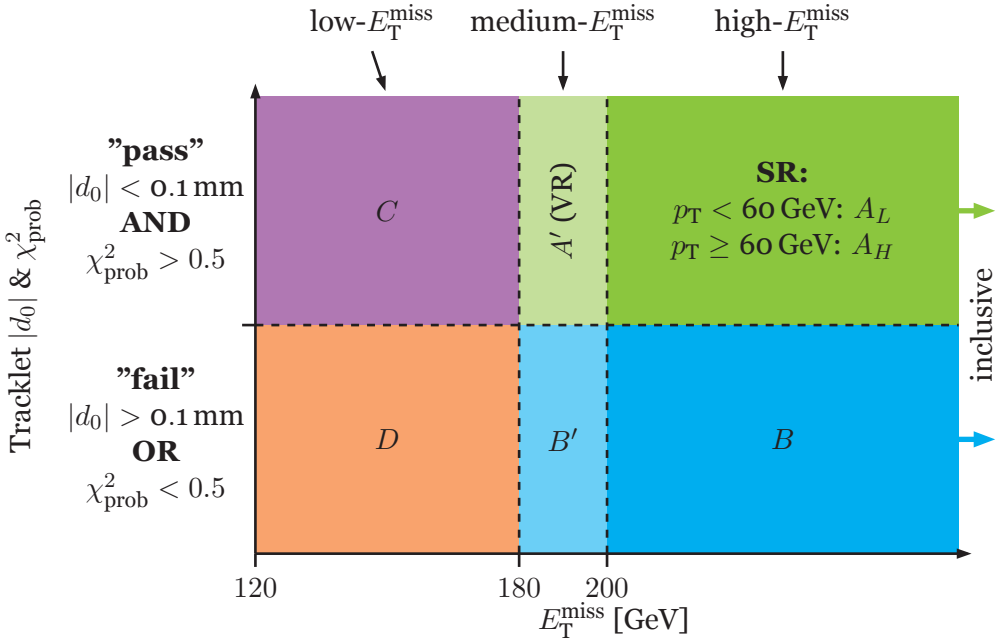


Figure 9.23.: Schematic of the ABCD plane used for the estimation of the combinatorial fake tracklet background. Two signal regions, two validation regions and two control regions are shown. They differ in their value of $E_{\mathrm{T}}^{\mathrm{miss}}$ and whether or not tracklets pass or fail a set of track based quality requirements. The signal region is subdivided into a low-p_{T} and a high-p_{T} region.

By using two of the auxiliary regions to calculate a transfer factor, and applying it to the

third, it is possible to obtain the background estimate as

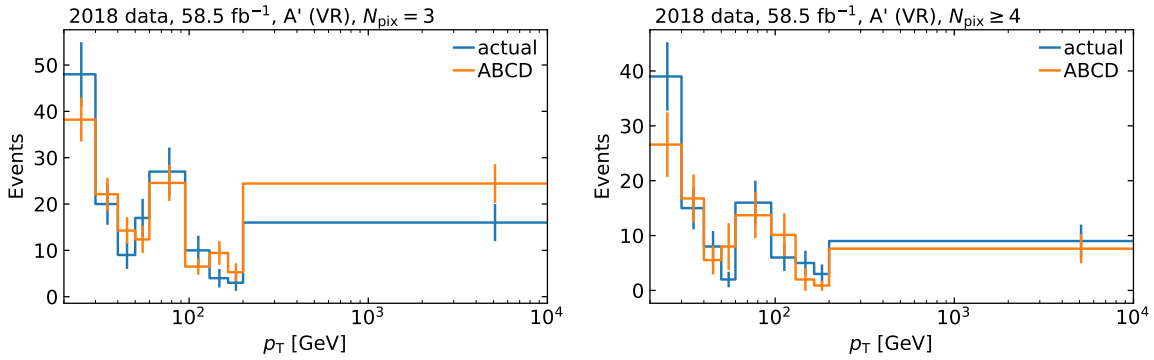
$$A_{\text{pred}} = \frac{C}{D} \times B \qquad \text{or} \qquad A_{\text{pred}} = \frac{B}{D} \times C. \tag{9.12}$$

The underlying assumption is that the physics transfers comparably from $C \to A$ and $D \to B$, or equivalently from $B \to A$ and $D \to C$. Additionally, the quantities used to define the ABCD plane need to be at most very weakly correlated, in order for the transfer relation to hold. This was verified to be the case for $E_{\text{T}}^{\text{miss}}$, $|d_0|$ and χ^2-probability as the defining quantities.

In addition, a validation region A' is defined as a slice in $E_{\text{T}}^{\text{miss}}$ between the signal region A and region C, in which the tracklet-quality criteria are passed. The sample of events below $E_{\text{T}}^{\text{miss}} < 200\,\text{GeV}$ is expected to have negligible signal contamination. A corresponding region B' is located in the tracklet-quality "fail" band between D and B, allowing an ABCD background estimate using

$$A'_{\text{pred}} = \frac{C}{D} \times B' \qquad \text{or} \qquad A'_{\text{pred}} = \frac{B'}{D} \times C, \tag{9.13}$$

for A'. This allows a comparison between background prediction and observed data in the validation region in order to evaluate the background estimate.



(a) $N_{\text{pix}} = 3$ (b) $N_{\text{pix}} \geq 4$

Figure 9.24.: Comparison of the p_{T} distribution of events in the A' validation region and the ABCD background prediction based on B'. Both tracklets with $N_{\text{pix}} = 3$ and $N_{\text{pix}} \geq 4$ are shown.

The inclusive ABCD method is found to result in a reasonable agreement between the background prediction and the actual events observed in the validation region. This can be seen in figure 9.24, which shows the predicted background p_{T} distribution and the actual distribution observed. The uncertainties on the actual validation region content are statistical, and uncertainties on the ABCD prediction are calculated from the statistical uncertainties on B, C and D. The binning was chosen to increase toward larger values of p_{T}, in order to reduce statistical uncertainties per bin. This results in the absolute bin content rising at 60 GeV and 200 GeV. While this binning is not strictly optimized, it was deemed appropriate and is used throughout the rest of this chapter. Within uncertainties, the background prediction agrees

with the data, indicating a reasonable background modelling.

9.9. Systematic uncertainties

Aside from statistical uncertainties due to the limited amount of events being analyzed, systematic effects can induce uncertainties in the analysis. As opposed to statistical uncertainties, these systematic effects do not necessarily decrease as more events are added to the analysis, and fluctuate around the correct value.

Determining which impact each systematic effect has depends strongly on the underlying sources of uncertainty. This section discusses the uncertainties that were identified as most relevant for the signal expectation and background estimate used in this analysis.

Simulated samples are typically generated by requiring pp-collisions to only result in one specific physics process. This is desirable in order to have a pure sample of the process, but results in an arbitrary effective integrated luminosity of the sample. Using the theoretical cross-section of the simulated process, and the integrated luminosity of the dataset that the simulated sample is compared with, it can be weighted to obtain an absolutely normalized set of events. The resulting sample, when applying the weights, then contains a total number of events with this specific process, that is expected to be contained in the dataset. The cross-section is calculated from theory, and carries a theoretical uncertainty, both of which are centrally provided. The luminosity is determined by ATLAS (see section 3.8), and its measurement is subjected to an uncertainty as well. The luminosity uncertainty only affects the normalization. As the overall signal expectation is a mixture of the processes $pp \to j\tilde{\chi}_1^{\pm}\tilde{\chi}_1^{\mp} + X$ and $pp \to j\tilde{\chi}^{\pm}\tilde{\chi}_{1,2}^{0} + X$, with differing cross-sections and uncertainties, the total uncertainty alters the shapes slightly.

Pile-up describes the condition where multiple inelastic interactions occur in a single proton-proton bunch crossing. It strongly affects the performance of reconstruction algorithms. The dataset used in the analysis has a fixed pile-up profile, which can be seen in figure 3.10. This pile-up profile has to be reproduced by simulated samples in order to be comparable. Ideally, samples are generated with the correct pile-up profile, by overlaying additional simulated events, and feeding them to the reconstruction chain alongside the process of interest. To improve pile-up profile modelling, a reweighting technique can be used. With these pile-up weights applied, the simulated profile should match the observed one. This reweighting comes with an associated uncertainty, that needs to be considered.

During jet reconstruction, calibration factors are applied so that jet energies correctly reproduce already measured resonance energies. This jet energy scale (JES) is determined using a combination of techniques, which each contribute to the overall calibration uncertainty. During calibration, jets in MC are also corrected such that their energy resolution (JER) matches the one observed in data, which introduces additional uncertainties. Analyses using jets in ATLAS are recommended to apply a vertex tagging requirement, which attempts to suppress jets coming from pile-up vertices. The efficiency of theses requirements differs between data and simulation which is corrected by a scale factor, with an associated uncertainty. In a similar

way, the modelling of the track soft term entering (TST) the $E_{\mathrm{T}}^{\mathrm{miss}}$ calculation in simulation and data are brought into agreement with a scale factor, which comes with an uncertainty.

Aside from the jet energy scale uncertainty, the aforementioned uncertainties exclusively apply to simulation, which is used for the signal expectation. Technically, the uncertainties are evaluated by varying the fundamental parameters associated with the uncertainty sources by $\pm 1\sigma$. Examples for this are trivial parameters like the luminosity scale factor, to dedicated sets of parameters provided centrally by the ATLAS performance groups. The event processing is then executed, and the effective impact on the selected samples is evaluated. By comparing the variations with the nominal selection, this can be quantified to some degree.

Table 9.9 shows the relative impact of the systematic uncertainties on the overall number of signal sample events entering the signal region, separated for three- and four-layer tracklet selections and different chargino masses. Individual fundamental parameter variation impacts are summed in quadrature to yield the combined numbers. The dominant uncertainties are found to be the cross-section and JES/JER related uncertainties. The $E_{\mathrm{T}}^{\mathrm{miss}}$ TST uncertainty is deemed to be negligible and is not considered further. Overall, systematic variations range from $-7.5\,\%$ up to $8.4\,\%$. Transverse momentum distributions of systematic variations of the signal expectation, and a comparison with the nominal expectations can be found in appendix B.1.

	$N_{\mathrm{pix}} = 3$			$N_{\mathrm{pix}} \geq 4$		
$m(\tilde{\chi}^{\pm})$	160 GeV	200 GeV	240 GeV	160 GeV	200 GeV	240 GeV
Cross-section	$\pm 3.7\,\%$	$\pm 4.1\,\%$	$\pm 4.5\,\%$	$\pm 3.7\,\%$	$\pm 4.2\,\%$	$\pm 4.4\,\%$
Pile-up modelling	$\pm 0.9\,\%$	$^{+0.4\,\%}_{-0.6\,\%}$	$^{+0.5\,\%}_{-0.9\,\%}$	$^{+0.5\,\%}_{-0.0\,\%}$	$^{+1.1\,\%}_{-1.0\,\%}$	$^{+0.0\,\%}_{-0.9\,\%}$
JVT efficiency	$\pm 0.3\,\%$	$\pm 0.3\,\%$	$\pm 0.4\,\%$	$\pm 0.3\,\%$	$\pm 0.3\,\%$	$\pm 0.3\,\%$
JES/JER	$^{+2.8\,\%}_{-6.2\,\%}$	$^{+1.4\,\%}_{-5.4\,\%}$	$^{+1.5\,\%}_{-2.3\,\%}$	$^{+1.3\,\%}_{-5.1\,\%}$	$^{+7.1\,\%}_{-1.5\,\%}$	$^{+4.4\,\%}_{-1.7\,\%}$
$E_{\mathrm{T}}^{\mathrm{miss}}$ track soft term	$^{+0.0\,\%}_{-1.2\,\%}$	$\pm 0.0\,\%$	$\pm 0.0\,\%$	$^{+0.0\,\%}_{-0.5\,\%}$	$\pm 0.0\,\%$	$\pm 0.0\,\%$
Luminosity	$\pm 2.0\,\%$					
Total	$^{+5.0\,\%}_{-7.5\,\%}$	$^{+4.9\,\%}_{-7.2\,\%}$	$^{+6.3\,\%}_{-6.5\,\%}$	$^{+4.0\,\%}_{-6.3\,\%}$	$^{+8.4\,\%}_{-4.7\,\%}$	$^{+6.7\,\%}_{-5.4\,\%}$

Table 9.9.: Impact of systematic uncertainties on signal expectation in the signal region. Each item is the root of the sum of squares of all variations associated with the group.

As the background prediction is entirely data driven, any uncertainties related to correcting simulation do not apply. The calibration of the jet energy scale is applied in data, and can therefore affect the observed distribution. Conventionally, systematic uncertainties on the selected data itself are suppressed, and the data is accepted as is with all systematics set to their nominal values.

As the background estimation is data driven, it is expected that most or all systematic effects affect the selected data and background estimate the same way. However, in order to be conservative, the jet energy scale uncertainty is evaluated and accounted for in the

interpretation of the data. This is done by varying the relevant fundamental parameters, and evaluating their impact on the ABCD background estimate.

An additional effect is evaluated for the background. As discussed before, the regions C, D are defined using intervals of 120 GeV $< E_{\mathrm{T}}^{\mathrm{miss}} <$ 180 GeV. The upper threshold is smaller then the signal region cut of 200 GeV to leave space for the validation region. Accordingly, regions A' and B' are defined as 180 GeV $< E_{\mathrm{T}}^{\mathrm{miss}} <$ 200 GeV. To quantify the stability of the background estimate, the lower and medium $E_{\mathrm{T}}^{\mathrm{miss}}$ thresholds are varied by 5 %, and the impact on the ABCD background estimate is evaluated. As the upper thresholds of 200 GeV is not changed, the signal region itself is not affected, and the varied background estimates can be directly compared with the nominal one.

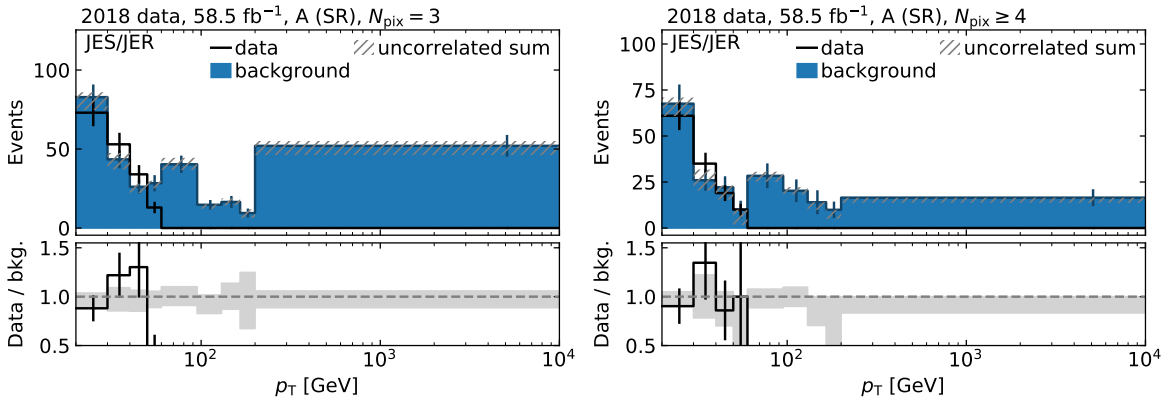
	$N_{\mathrm{pix}} = 3$	$N_{\mathrm{pix}} \geq 4$
JES/JER	$+4.9$ % -10.9 %	$+1.8$ % -14.9 %
ABCD $E_{\mathrm{T}}^{\mathrm{miss}}$ threshold	$+1.1$ % -0.6 %	$+1.7$ % -5.4 %
Total	$+5.1$ % -10.9 %	$+2.5$ % -15.9 %

Table 9.10.: Impact of systematic uncertainties on the background estimate derived via ABCD. Each item is the root of the sum of squares of all variations associated with the group.

Table 9.10 shows the relative impact of both the jet energy scale related uncertainties as well as the variation of the $E_{\mathrm{T}}^{\mathrm{miss}}$ thresholds. The former are still labeled as *JES/JER*, since technically the fundamental parameters related to the jet energy resolution are also varied. As the resolution correction factors are not actually applied in data, however, these variations are ignored. Three- and four-layer background estimates are listed separately. Overall, the background estimate has relative systematic uncertainties of 5.1 % and -10.9 %.

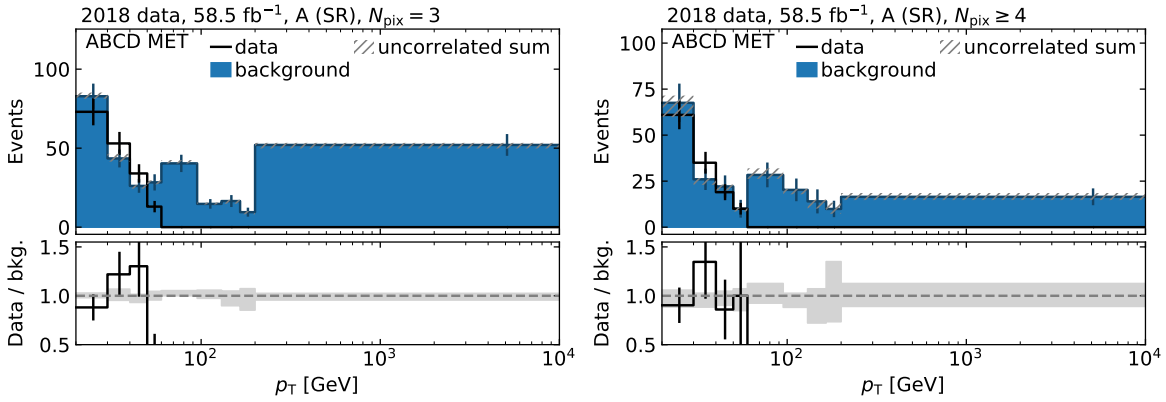
Figures 9.25 and 9.26 show p_{T} distributions in the signal region and indicate the jet energy scale and ABCD $E_{\mathrm{T}}^{\mathrm{miss}}$ threshold uncertainties, respectively. Both three- and four-layer distributions are displayed. Note that above $p_{\mathrm{T}} >$ 60 GeV the signal region is blinded, therefore bins above this value are empty in data. The lower plot panels contain the ratio of data and the background prediction, and the ratio between the systematic variations and the nominal background estimate. Individual uncertainty variations are again summed in quadrature. It can be seen that the impact of the respective variations is relatively constant across the p_{T} range, while some fluctuations can be observed.

Equivalent plots for the validation region can be found in appendix B.1. Here, the lower statistics lead to much larger fluctuations. Also, since the ABCD $E_{\mathrm{T}}^{\mathrm{miss}}$ threshold variations directly influences the definition of the validation region itself, its impact on the background is very large.



(a) (b)

Figure 9.25.: Transverse momentum distributions in the signal region for three- and four-layer tracklets. Also shown are the nominal background estimate, as well as statistical and systematic uncertainties related to the jet energy scale. The ratio between data and background, as well as its uncertainty are given in the lower panel.



(a) (b)

Figure 9.26.: Transverse momentum distributions in the signal region for three- and four-layer tracklets. Also shown are the nominal background estimate, as well as statistical and systematic uncertainties related to the ABCD E_T^{miss} threshold choice. The ratio between data and background, as well as uncertainty are given in the lower panel.

9.10. Results and statistical interpretation

This section contains details on the statistical interpretation of the results of this analysis. First, a brief introduction into the likelihood-based approach used is given, before its application and final results are discussed.

9.10.1. Likelihood-based hypothesis tests

The purpose of the statistical analysis is to interpret the obtained results and quantify them. In case of observing a signal, it is used to determine the degree of certainty associated with that observation in the form of a significance value. In the absence of a signal, it can be used to extract upper limits on the amount of signal statistically compatible with the observation. These limits can also be converted into different representations, such as exclusion limits on model parameters. In the case of this analysis, model parameters of interest are the chargino mass and lifetime.

Different approaches exist to perform a statistical interpretation. At the highest level, they can be grouped into two distinct classes: *Bayesian* and *frequentist*. The interpretation of a *probability* differs between the two classes. Bayesian statistics builds on Bayes' theorem

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}, \tag{9.14}$$

which relates the conditional probability of an event A given event B to the opposite conditional probability and individual probabilities. It can be used to determine the probability of an observation given a certain hypothesis. In context of an analysis like the one presented in this thesis, A can be identified with the signal hypothesis, while B represents the observed data set. Given B, the probability of the signal hypothesis can be quantified. Bayesian statistics interprets probabilities as the *degree-of-belief* for a specific condition. It also requires the choice of prior probabilities, such as $P(A)$ in equation (9.14), which are typically hard to motivate.

The second major class is frequentist statistics. At its core is the interpretation of a probability as the fraction of occurrences of specific events, in the limit of infinite trials. Therefore, probabilities are considered purely as statements on the expected frequencies of events. Also, prior probabilities are not required for frequentist calculations.

As described in [238], frequentist principles can be employed to conduct a statistical test for a search for new physics. A hypothesis test is performed, where a null hypothesis is associated with only known processes, and a signal hypothesis contains the new phenomena. The SUSY model under study in this analysis is therefore associated with the signal hypothesis. In such a hypothesis test, the frequentist approach results in a statement on the probability for the observed data, assuming the background-only hypothesis. The contrary probability of said hypothesis given the data is a purely Bayesian statement.

Starting from a histogram of measured events with N bins

$$\vec{n} = (n_1, \ldots, n_N), \tag{9.15}$$

the expectation value of n_i is parametrized as

$$E[n_i] = \mu s_i + b_i, \tag{9.16}$$

with the nominally expected number of signal events s_i, the estimated background events b_i,

and the *signal strength* μ which scales the signal expectation. Each bin content is the integral over the corresponding probability density functions of the quantities:

$$s_i = s_{\text{tot}} \int_{\text{bin } i} f_s(x, \vec{\theta}_s)\, \mathrm{d}x \tag{9.17}$$

$$b_i = b_{\text{tot}} \int_{\text{bin } i} f_b(x, \vec{\theta}_b)\, \mathrm{d}x. \tag{9.18}$$

It depends on the histogram quantity itself and a set of Nuisance Parameters (NPs) $\vec{\theta}_{s,b}$. b_{tot} is the overall background normalization, that can optionally also be included as an NP: $\vec{\theta} = (\vec{\theta}_s, \vec{\theta}_b, b_{\text{tot}})$. For this analysis, the normalization of the background is not determined in the fit, therefore b_{tot} is dropped. NPs can be used to incorporate uncertainties into the interpretation, such as the systematics uncertainties discussed in section 9.9.

A test statistic is needed to define the p-values that enter the hypothesis test. Following the *Neyman-Pearson lemma* [239], a likelihood ratio is used as the basis of the test statistic. The first ingredient is the likelihood function, which can be defined as the product of Poisson probabilities of the individual N histogram bins:

$$L(\mu, \vec{\theta}) = \prod_{j=1}^{N} \frac{(\mu s_j + b_j)^{n_j}}{n_j!} e^{-(\mu s_j + b_j)} \prod_{k=1}^{M} \frac{u_k^{m_k}}{m_k!} e^{-u_k}. \tag{9.19}$$

The second term includes auxiliary measurement histograms m_k, which can be used to constrain the NPs, using measurable quantities $u_k(\vec{\theta})$. For this analysis, the A_L region is treated as an auxiliary measurement, while A_H is treated as the search histogram n_i.

For a specific value of μ, the *profile-likelihood ratio* is:

$$\lambda(\mu) = \frac{L(\mu, \hat{\hat{\vec{\theta}}})}{L(\hat{\mu}, \hat{\vec{\theta}})} \tag{9.20}$$

Here, $\hat{\hat{\vec{\theta}}}$ is the value of $\vec{\theta}$ that maximized the likelihood function $L(\mu, \vec{\theta})$ for a specific fixed μ. $L(\hat{\mu}, \hat{\vec{\theta}})$ is called the unconditional likelihood function, where both $\hat{\mu}$ and $\hat{\vec{\theta}}$ are determined by maximizing the likelihood function. Finally, for the purpose of determining an upper limit on μ, a one sided test statistic is defined as a variation of the profile-likelihood ratio:

$$\tilde{q}_\mu = \begin{cases} -2 \ln \frac{L(\mu, \hat{\hat{\vec{\theta}}}(\mu))}{L(0, \hat{\hat{\vec{\theta}}}(0))}, & \hat{\mu} < 0, \\ -2 \ln \frac{L(\mu, \hat{\hat{\vec{\theta}}}(\mu))}{L(\hat{\mu}, \hat{\vec{\theta}})}, & 0 \leq \hat{\mu} \leq \mu, \\ 0, & \hat{\mu} > \mu. \end{cases} \tag{9.21}$$

This test statistic is chosen such that negative values of $\hat{\mu}$ are divided by the likelihood extracted at $\mu = 0$. Additionally, values of μ smaller than the maximum likelihood value of $\hat{\mu}$ are truncated to zero.

Using the probability distribution function $f(\tilde{q}_\mu|\mu)$ of the test statistic, the associated p-value of an observed set of n_i leading to $\tilde{q}_{\mu,\text{obs}}$ is defined as

$$p_\mu = \int_{\tilde{q}_{\mu,\text{obs}}}^{\infty} f(\tilde{q}_\mu|\mu)\,\mathrm{d}q_\mu \overset{!}{=} 1-\alpha, \tag{9.22}$$

with the Confidence Level (CL) α. Typically, a CL of $\alpha = 0.95$ is used. An illustration of the relationship between the probability density function of the test statistic and the observed value of $\tilde{q}_{\mu,\text{obs}}$ for the data n_i is given in figure 9.27. It also shows the p_μ-value as the integral from $\tilde{q}_{\mu,\text{obs}}$ to infinity.

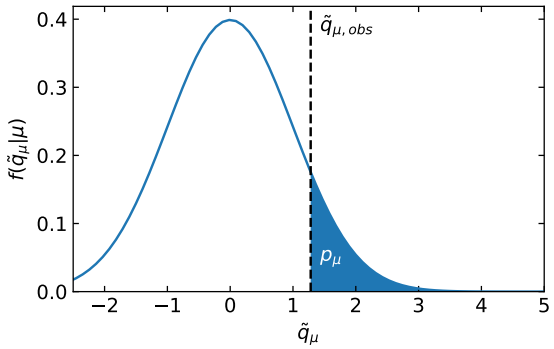


Figure 9.27.: Illustration of the relationship of the probability density function $f(\tilde{q}_\mu|\mu)$, $\tilde{q}_{\mu,\text{obs}}$ and the p-value p_μ. The shaded area represents the integrated probability corresponding to a chosen Confidence Level of 90 %, as an example.

Rather than directly using this p-value, the so-called CL_s method [240] is used. Here, the value CL_s is defined as

$$\text{CL}_s = \frac{p_{s+b}}{p_b}, \tag{9.23}$$

the ratio of the p-value under the assumption of the signal hypothesis $s+b$ ($\mu = 1$) over p_b, the p-value under the background-only hypothesis ($\mu = 0$). Figure 9.28 shows an illustration of these p-values with respect to the underlying probability density functions $f(\tilde{q}_\mu|s+b)$ and $f(\tilde{q}_\mu|b)$. The CL_s method has the advantage of correctly covering the threshold of zero signal. Unlike the direct approach, downward fluctuations are not automatically interpreted as evidence against a low μ signal, but rather as insufficient sensitivity in that parameter region.

For the extraction of the p-values discussed above, the probability density functions of the test statistic are a critical input. As it is not available analytically, it is typically derived in one of two ways. The first approach uses so-called *toy* experiments. In these, the background estimate is varied according to its statistics, while setting all NPs to their maximum-likelihood values as determined from a fit to data. For each variation, the test statistic is evaluated at $\mu = 0$ and $\mu = 1$. The resulting ensemble of $\tilde{q}_\mu$ yields the distribution of the test statistic. The generation of sufficiently large toy-samples is often very expensive computationally. As

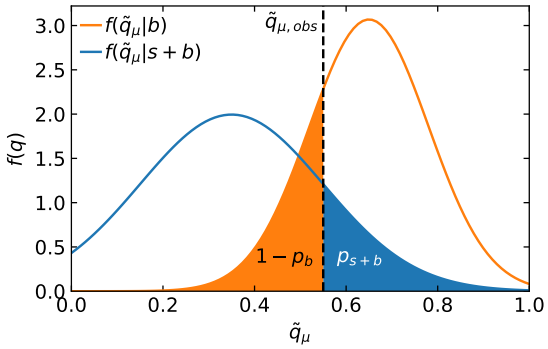


Figure 9.28.: Illustration of the way separate p-values p_{s+b} and p_b are combined to form the final CL_s value. The value range of q is arbitrary for the purpose of the illustration.

described in [238], approximate methods exist that use an asymptotic approach to estimate the test statistic distribution. The statistical analysis presented here uses this asymptotic approximation exclusively.

For the signal model under study in this thesis, upper limits on the signal strength μ are calculated. By themselves, they quantify the maximum amount of signal that would have to be present to be incompatible with observed data at a chosen CL. It is calculated by scanning the a series of μ values. At each step, the CL_s value is derived. The range of μ values needs to be large enough such that this p-value drops below $1-\alpha = 0.05$, corresponding to a CL of 95 %. Subsequently, the μ value for which $CL_s = 5\,\%$ is extracted either by calculating additional test points, or by linear interpolation, yielding the upper limit on the signal strength μ_{up}. An illustration of the μ-scan used to obtain the upper limit is shown in figure 9.29. The p-value starts out at q, and begins to drop as μ increases. The crossover point below 0.05 is marked, indicating the value of μ_{up}.

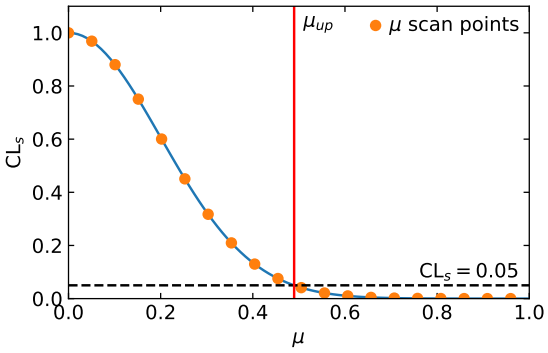


Figure 9.29.: Illustration of a μ-scan used to determine the upper limit on the signal strength parameter, using the CL_s method at a CL of 95 %.

The upper limit can be calculated with observed data, yielding the *observed* limit. Alternatively, the background estimate can be used in its place, after a maximum likelihood fit

to data, to constrain the nuisance parameters. This yields the *expected* upper limit, which is useful to quantify what the expected sensitivity for exclusion of a signal model is, without actually looking at the observed data.

The implementation of the statistical analysis, whose results are shown in the subsequent section uses the ATLAS internal `TRExFitter` library, which uses the `HistFactory` software and corresponding formalism [241]. It describes the convention on how to construct the likelihood function that is at the center of this statistical analysis approach. It can consume inputs in the form of histograms, and is able to extract constrained systematics distributions by interpolating between the up and down variations shown in section 9.9. It conducts the maximum-likelihood fits necessary to determine the parameters of the test statistic. One result are the *post-fit* versions of the signal and background histograms, which can be compared to the data to obtain an understanding of the way the fit made use of the systematic uncertainties. The software is also able to extract upper limits on the signal strength, both observed and expected. In the latter case, it also provides $\pm 1\sigma$ and $\pm 2\sigma$ uncertainty boundaries.

9.10.2. Expected signal sensitivity

Fit results

Using the formalism outlined in the previous section, the statistical analysis of the disappearing track search was performed. In order to be conservative and not influence optimization of the analysis, the data remained blinded in the signal region A_H. To compensate for this, and to still be able to perform the required maximum likelihood fits, a set of pseudo-data was generated from the background estimate in A_H. This was done by randomly sampling a Gaussian distribution with a central value equal to the bin content, and a variance equal to the variance of the bin. A Gaussian distribution is used rather than a Poisson distribution, since the background estimate is scaled using the ABCD factors, and the bin variance is affected by the statistics used in the ABCD calculation. A Poisson distribution would fix the variance based on the bin value, thereby not allowing correct propagation of the ABCD related statistical uncertainty. In the low-p_{T} region, actual data is used. With this combined dataset of real data and pseudo-data, the limit setting procedure is executed and upper limits on the signal strength μ are calculated. This section presents the expected limits, in order to study the achievable sensitivity with the unified background estimation.

Figure 9.30 shows transverse momentum distributions in the A_H region, that are the input to the fitting and limit setting calculation. Both three- and four-layer selections are displayed separately. The background estimate is shown in blue, while the pseudo-data generated from it is shown in orange. An example of a signal hypothesis with a chargino mass of 240 GeV and a lifetime of 0.1 ns is shown in green. In the pre-fit distributions, this hypothesis is clearly visible, while in the post-fit distributions it vanishes. This is expected for a fit to the pseudo-data, which should not accommodate a residual signal. The lower panel shows the ratio between the pseudo-data and the signal plus background expectation. In the presence of signal in the pre-fit distribution, this ratio is low. Post-fit, it moves much closer to unity, as expected.

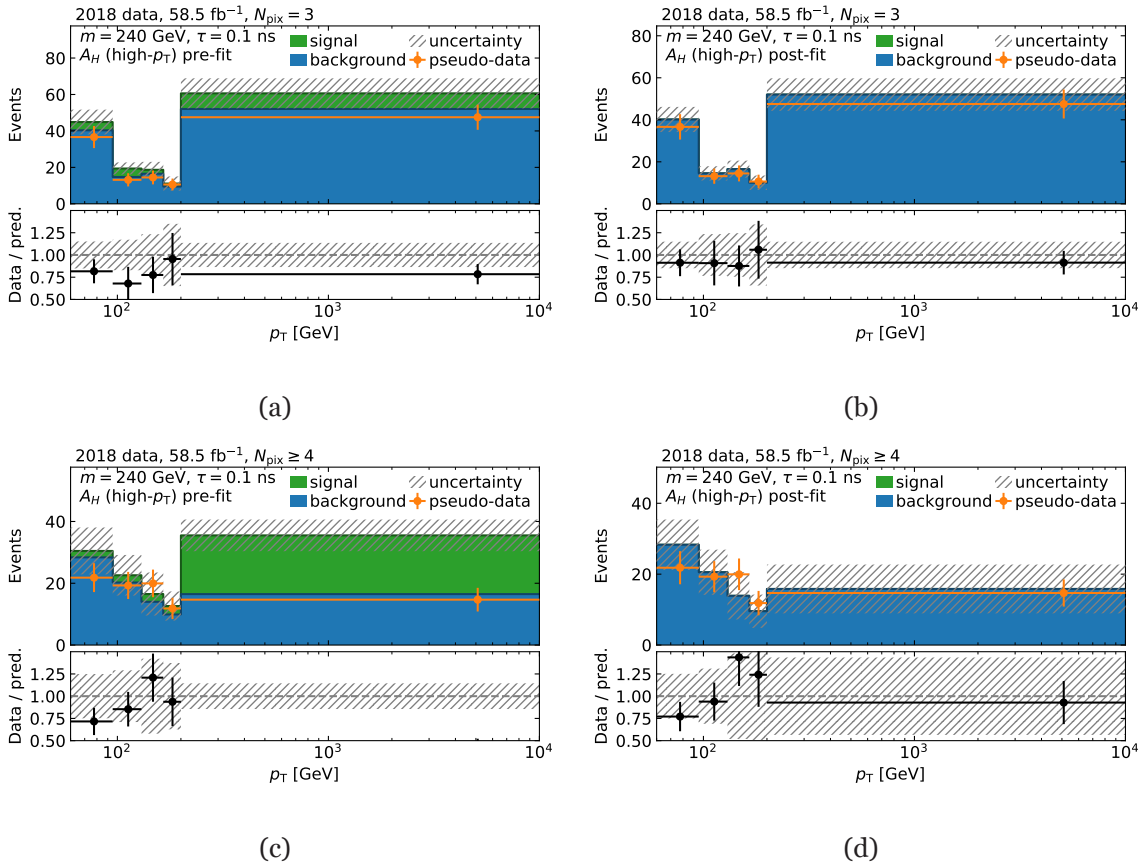


Figure 9.30.: Examples of combined pre- and post-fit p_T distributions for three- and four-layer tracklet selections in the A_H region. The fit is performed simultaneously to three- and four-layer distributions. The background estimate is shown, as is the data and generated pseudo-data. A signal hypothesis for $m = 240$ GeV, $\tau = 0.1$ ns and $\mu = 1$ is shown. The post-fit distributions have small values of μ, rendering the signal hypothesis invisible. The lower panel shows the ratio between the pseudo-data and the signal plus background expectation. Absolute and relative statistical and systematic uncertainties are shown as well.

Statistical uncertainties of the pseudo-data and statistical and systematic uncertainties for the background and signal are shown. The relative uncertainty at high-p_T increases from pre- to post-fit, due to the signal disappearing. Overall, the systematic uncertainties are around ± 25 % to ± 40 %, which is relatively large. In the interest of being conservative, this is acceptable. Additional examples of p_T distributions with signals at other lifetimes are located in appendix B.2.

In order to validate the overall background estimation and fitting procedure, the A_L region can be used. As it is expected to be nearly signal free, real data is shown here, and can be compared to the background estimate. Figure 9.31 shows transverse momentum distributions in the A_L region, pre- and post-fit. The data and background estimate is shown, as is the signal hypothesis from before, which is negligible in magnitude. Distributions for three- and four-layer selections are visible. The lower panels indicate the ratio between the data and

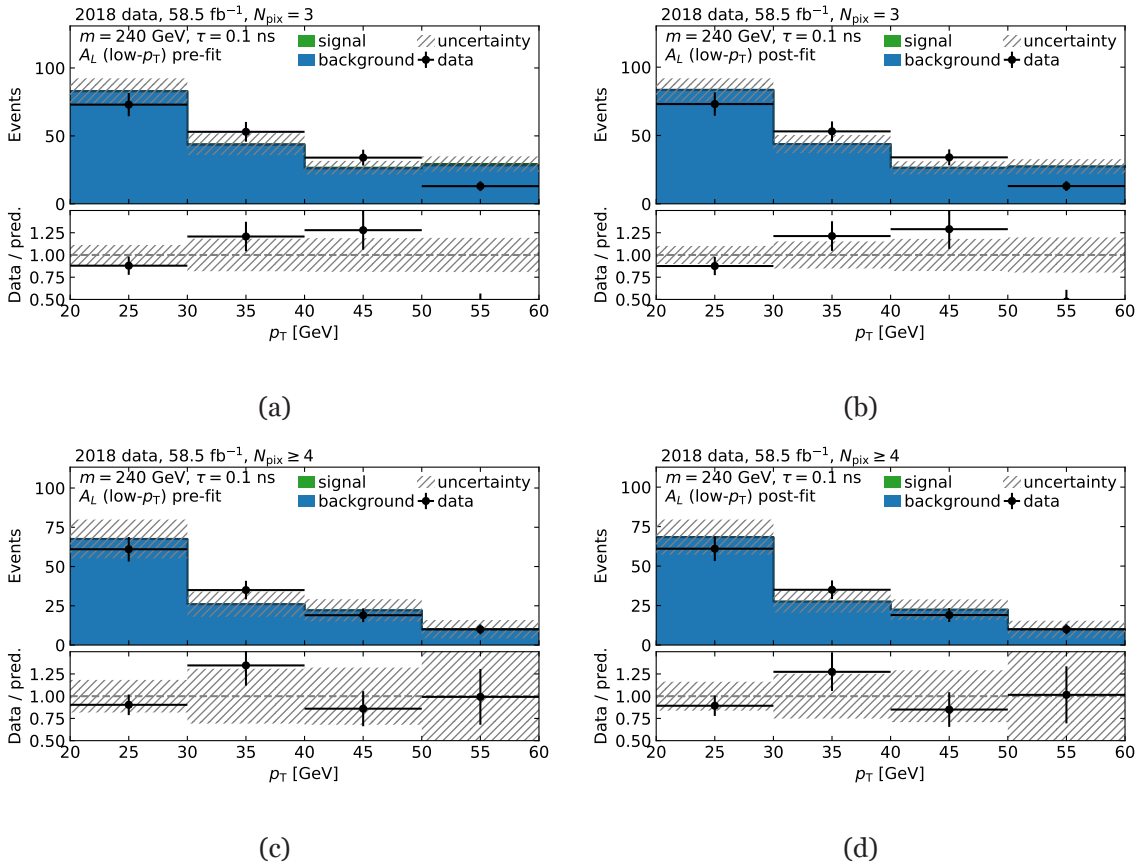


Figure 9.31.: Examples of combined pre- and post-fit p_T distributions for three- and four-layer tracklet selections in the A_L region. The fit is performed simultaneously to three- and four-layer distributions. The background estimate is shown, as is the data. A signal hypothesis for $m = 240$ GeV and $\tau = 0.1$ ns is shown, but is negligibly small. The lower panel shows the ratio between the data and the signal plus background expectation. Absolute and relative statistical and systematic uncertainties are shown as well.

background estimate. Absolute and relative statistical and systematic uncertainties are shown. The agreement between background and data is better for four-layer tracklets than it is for three-layer tracklets. The data histogram fluctuates strongly, which is partially covered by its statistical uncertainty. Uncertainties on the background estimate cover most of the observed discrepancies, with the exception of the highest-p_T bin for the three-layer distributions. The observed tension between data and background estimates indicates the limitations of the unified background estimation approach that is presented here.

Post-fit pull behavior of nuisance parameters was investigated to ensure a reasonably stable fit. A visualization of nuisance parameter pulls can be seen in figure 9.32 for $m(\tilde{\chi}_1^{\pm}) = 240$ GeV and $\tau(\tilde{\chi}_1^{\pm}) = 0.1$ ns. The horizontal axis shows the pull, meaning the residual between the post-fit value and the nominal one, divided by the parameter uncertainty. Error bars indicate the uncertainty on the parameter calculated from the fit. Labels on the right indicate the technical NP name, which can be used to categorize them. No NPs were found to be pulled

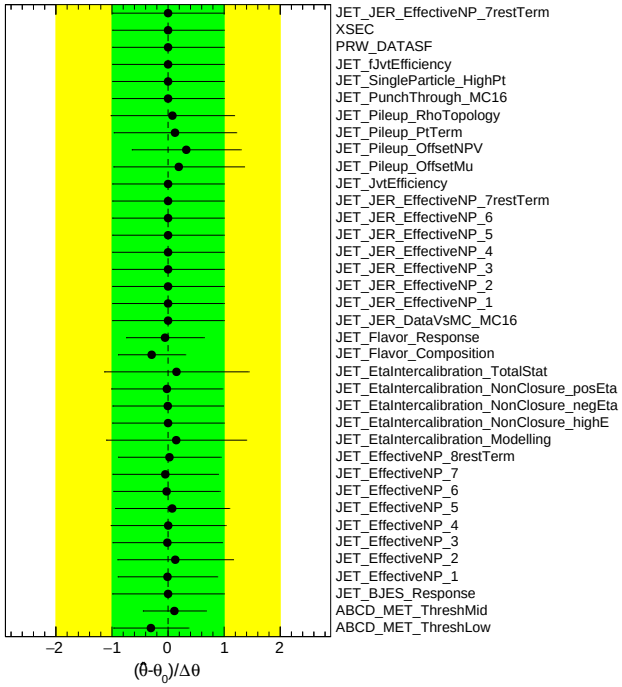


Figure 9.32.: Plot showing the nuisance parameter pulls after a maximum likelihood fit. A signal sample with $m(\tilde{\chi}_1^{\pm}) = 240$ GeV and $\tau(\tilde{\chi}_1^{\pm}) = 0.1$ ns is used. Green and yellow bands indicate the $\pm 1\sigma$ and $\pm 2\sigma$ widths, while error bars show the parameter uncertainty obtained from the fit.

strongly, in line with the expectation for a fit to pseudo-data. Some NP showed a reduced uncertainty post-fit, which would naively be identified by a physical constraint being made by the observed data. In a fit to pseudo-data, this is not expected. With the exception of one NP, this only occurs for the NPs associated with the ABCD $E_{\mathrm{T}}^{\mathrm{miss}}$ threshold uncertainty. In this case, the fit in fact *learns* the nominal position of the $E_{\mathrm{T}}^{\mathrm{miss}}$ threshold by variation of the background estimate, which explains the constraint. Further investigation would be needed to fully understand the NP behavior. Additional examples of NP pull visualizations can be found in appendix B.3.

Signal strength upper limits

Upper limits are calculated both for three- and four-layer tracklet selections individually, in addition to a combined limit, which considers both selections simultaneously. In the combined fit, all NPs are treated in a correlated fashion between the two selections. The calculation is performed for three chargino mass points $m(\tilde{\chi}_1^{\pm}) \in \{160, 200, 240\}$ GeV. Additionally, a series of signal templates for lifetimes between $\tau(\tilde{\chi}_1^{\pm}) = 0.04$ ns and 0.2 ns were generated via the weighting approach showed in section 9.4.2.

Figure 9.33 shows the result of the upper limit calculation for an lifetime of 0.0575 ns at 95 % CL. It shows the expected upper limit μ_{up} from the combined three- and four-layer fit as a dashed black line. Also shown are the $\pm 1\sigma$ and $\pm 2\sigma$ bands in green and yellow, respectively.

Solid orange and red lines depict the three-layer-only and four-layer-only upper limits for comparison. A horizontal dashed line indicates $\mu = 1$. Upper limits above this point cannot exclude a signal process with the defined characteristics at the stated CL, as the nominal signal strength is still statistically compatible with the background estimate. This means that the analysis has insufficient sensitivity at this mass and lifetime. By interpolating the intersection point between the expected limits and the $\mu = 1$ line, a mass exclusion limit can be calculated. In the example shown this is found to be about $m > 199\,\text{GeV}$. The interpretation of this number is that in the given analysis, a chargino with the stated lifetime that is below this mass threshold is expected to yield a signal that should be visible. An equivalent interpretation is that given the analysis, the signal hypothesis can be rejected for masses below this threshold. Note that this intersection does not strictly yield a true 95 % CL lower mass limit, but is an approximation. Examples of upper limits on the signal strength for additional lifetimes can be found in appendix B.4.

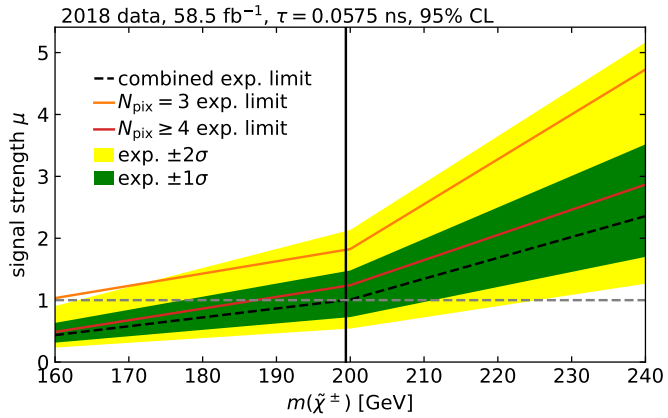


Figure 9.33.: Example of a set of upper limits on the signal strength, at a lifetime of $\tau = 0.0575\,\text{ns}$ at 95 % CL. Shown are the expected limit and $\pm 1\sigma$ and $\pm 2\sigma$ bands for the combined limit. Individual upper limits for three- and four-layer tracklet selections are also shown.

A complete analysis would complement this expected exclusion limit with an observed limit, using the data itself, and comparing it with the expected limit. As this requires unblinding the data, and a very thorough understanding of all details of the analysis strategy is needed to confidently unblind, this thesis only shows expected limits.

It was found that the limit calculation shows technical failures for low lifetimes. For four-layer tracklets, these failures manifest at around $\tau \approx 0.05\,\text{ns}$, while for three-layer tracklets it is stable until $\tau \approx 0.045\,\text{ns}$. The combined limit calculation succeeds in the entire lifetime range down to 0.04 ns. Further development effort would have to be committed to this problem to understand its nature and resolve it. However, as mass exclusion limits reach the lower bound of the available mass range above critical lifetimes, the failing lifetimes are not required to make statements about the sensitivity of the method.

Mass exclusion limits

It is instructive to convert the upper limits shown in the previous section to lower limits on the mass. As stated above, this conversion is achieved by finding the mass at which the signal strength upper limit $\mu_{\text{up}} = 1$. Signals with masses below this value are expected to be excluded, as they would result in signature the analysis is sensitive to. This section presents these mass exclusions as contours in the mass-lifetime plane. For each combination of lifetime and mass, the mass exclusion point is extracted, and visualized. Note that test signal samples were only available for a mass range of 160 GeV $\leq m(\tilde{\chi}_1^{\pm}) \leq$ 240 GeV, restricting the lifetime range to one where mass exclusion falls within these masses.

Figure 9.34 shows the expected exclusion lines determined in this way from three- and four-layer signal strength limits, as well as a combined limit, for 58.5 fb^{-1} of integrated luminosity from 2018 data. The dashed colored lines indicate the connection between the minimum and maximum intersection observed within the available signal mass range, and the first lifetime where the intersection falls outside of the range. No explicit mass bound can be calculated outside of this range. Also shown is the theoretically expected lifetime for a pure-higgsino scenario, where the mass splitting is generated exclusively by radiative corrections ([229], see section 9.2). It is clear that in order to improve the exclusion of a pure-higgsino signal model, mass exclusion limits need to be pushed toward lower lifetimes. This, in turn, would move the intersection point between the limit line and the theoretical expectation toward higher masses, thereby increasing the signal exclusion at larger masses. The intersection between the combined lower mass bound and the theoretical lifetime curve is found at just above 160 GeV, the lower end of the available signal sample mass range.

It can be observed that separately, the three-layer limit is considerably weaker than the four-layer limit. Lowering the required number of contributing Pixel layers will likely increase the background entering the signal selection. In particular the background from random combinations of measurements is expected to play a larger role at only three Pixel layers.

The combination of three- and four-layer tracklets achieves an improvement over the four-layer standalone limit. As a point of reference, the expected exclusion line from the most recent publication on a search for pure-higgsino signals [229] is shown as well. The analysis in this publication differs substantially from the one presented in this thesis. It exclusively uses four-layer tracklets, and employs a background estimation strategy that is conceptually closer to the fit normalization approach discussed in section 9.7. The event selection strategy differs in a number of of aspects, while the tracklet selection in this analysis was largely modelled in line with the publication, and cut values are similar. The four-layer expected limit shows a small improvement with respect to the published result, while the three-layer limit appears to trail in sensitivity.

Aside from the discrepancies mentioned before, the published result uses a dataset amounting to 36.1 fb^{-1} of integrated luminosity from 2015 and 2016 data taking. To enable a more direct comparison, the dataset and signal sample used for this analysis was scaled down to this luminosity, the result of which is visible in figure 9.35. Note that even with scaled luminosity, differences remain. As discussed in chapter 8, the tracklet reconstruction was

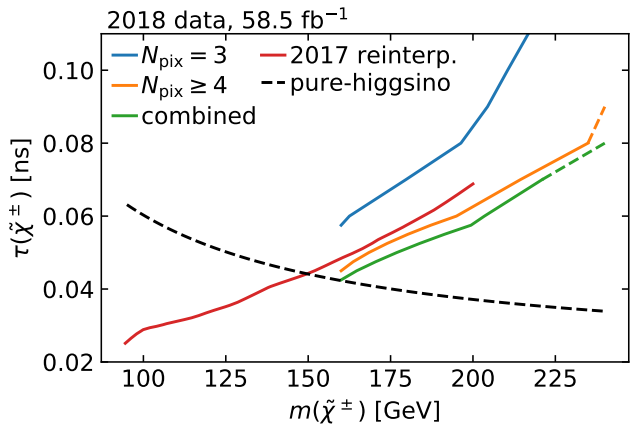


Figure 9.34.: Expected mass exclusion limits based on 95 % CL signal strength upper limits, shown in the plane of the chargino mass and lifetime. Limits obtained from only three-layer and four-layer tracklets, as well as a combination are drawn. Also shown is the expected 95 % CL limit obtained in the higgsino reinterpretation from [229] using 36.1 fb^{-1} of data, and the theoretical pure-higgsino lifetime.

adapted and includes a special SCT hit-filter. This filter improves the signal reconstruction efficiency, and was not present in the reconstruction used for the publication. On the other hand, pile-up increased from 2015+2016 to 2018 (see figure 3.10), which is expected to have a negative effect on the amount of selected background. The magnitude of both of these effects are either small, or approximately cancel each other, as the scaled expected mass exclusion line for four-layers derived in this thesis closely matched the published line. Again, the standalone three-layer limit is not competitive in sensitivity, whereas the combination yields a comfortable improvement with respect to the published result.

Both the published result as well as the tracklet selection discussed in section 9.5.1 are not strongly optimized toward the improved tracklet reconstruction and inclusion of three-layer tracklets. Apart from the changed minimum Pixel layer requirement, no specific optimization has been conducted. To evaluate what the expected improvement of such an optimization could be, the $|z_0 \sin\theta|$ requirement was tightened from 0.5 mm to 0.1 mm. The expectation is that this should assist in suppression of tracklets from random combinations, as these do not favor low absolute impact parameter values. As three-layer tracklets are more strongly affected by the fake background, the improvement in this channel is expected to be larger.

Figure 9.36 shows the resulting expected mass exclusion limits for the tightened $|z_0 \sin\theta| <$ 0.1 mm cut. As is clearly visible, the standalone three-layer limit improves strongly, and matches the four-layer limit up to about $m = 200$ GeV. An interpretation of this observation is that the tighter impact parameter cut helps reduce the fake background, and brings the background estimate in line with the four-layer tracklets case. As a consequence, the combined limit shows a strong improvement over either of the standalone limits, as well as the published result. This indicates that further work to improve the background estimate, uncertainty determination and limit extraction is worthwhile in achieving improved sensitivity at low

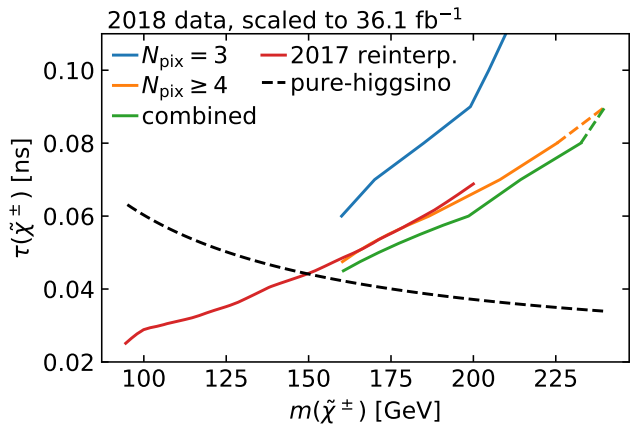


Figure 9.35.: Expected mass exclusion limits based on 95 % CL signal strength upper limits, shown in the plane of the chargino mass and lifetime. The underlying distributions were scaled to 36.1 fb^{-1}. Limits obtained from only three-layer and four-layer tracklets, as well as a combination are drawn. Also shown are the expected 95 % CL limit obtained in the higgsino reinterpretation from [229] using 36.1 fb^{-1} of data, and the theoretical pure-higgsino lifetime.

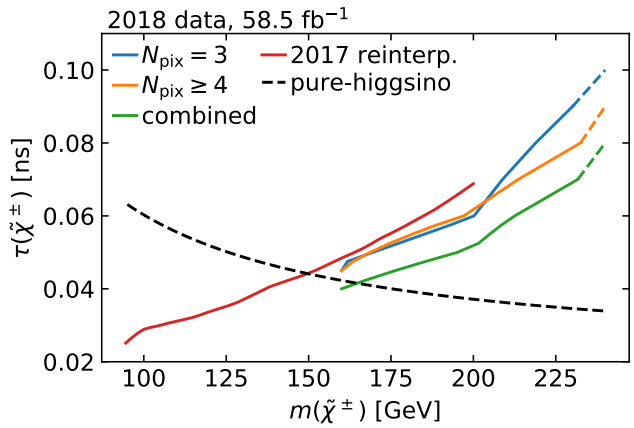
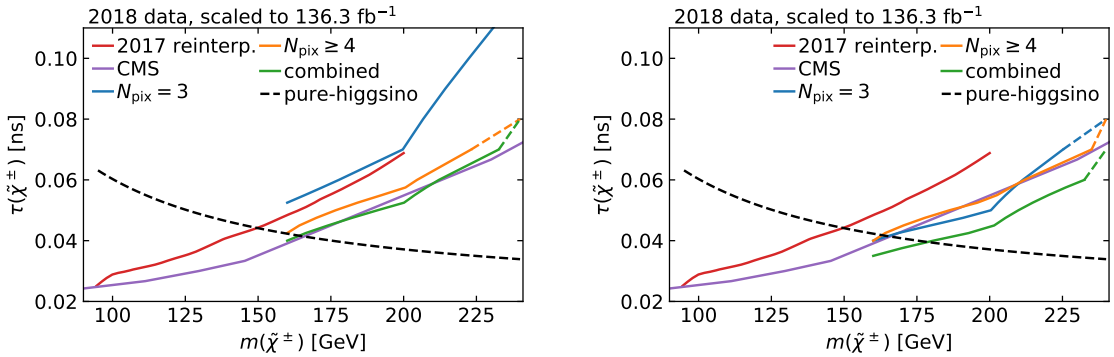


Figure 9.36.: Expected mass exclusion limits based on 95 % CL signal strength upper limits, shown in the plane of the chargino mass and lifetime, and using a tightened cut $|z_0 \sin\theta| < 0.1$ mm. Limits obtained from only three-layer and four-layer tracklets, as well as a combination are drawn. Also shown are the expected 95 % CL limit obtained in the higgsino reinterpretation from [229] using 36.1 fb^{-1} of data, and the theoretical pure-higgsino lifetime.

lifetimes for searches for a pure-higgsino signal scenario.

Finally, figure 9.37 contains equivalent expected limits lines, but scaled to the full LHC Run 2 integrated luminosity of 136.3 fb^{-1} that is usable for disappearing track searches. Also shown is the same scaled limit, but with the tightened impact parameter cut $|z_0 \sin\theta| < 0.1$ mm. As a comparison, the expected limit obtained by CMS using the full Run 2 dataset [222] is included as well.



(a) $|z_0 \sin\theta| < 0.5\,\text{mm}$ (b) $|z_0 \sin\theta| < 0.1\,\text{mm}$

Figure 9.37.: Expected mass exclusion limits based on 95 % CL signal strength upper limits, shown in the plane of the chargino mass and lifetime. The underlying distributions were scaled to 136.3 fb^{-1}, which is the full Run 2 ATLAS luminosity usable for disappearing track searches. Limits obtained from only three-layer and four-layer tracklets, as well as a combination is drawn. Also shown are the expected 95 % CL limit obtained in the higgsino reinterpretation from [229] using 36.1 fb^{-1} of data, the CMS full Run 2 expected limit [222], and the theoretical pure-higgsino lifetime.

It is sensible to compare these expected exclusion limits with the published CMS pure-higgsino results with the full Run 2 dataset [222], in the absence of an equivalent ATLAS publication. CMS quotes the observed pure-higgsino mass exclusion limit at $\tau = 0.05$ ns to be 175 GeV. At this lifetime and scaled luminosity, the projection for expected mass exclusion based on the analysis presented here are 191 GeV and 210 GeV for the nominal and tightened impact parameter cut using both three- and four-tracklet layer selections. At the nominal pure-higgsino lifetime, the expected mass limit found[10] by CMS is 166 GeV. The projections based on the analysis presented here are 165 GeV and 179 GeV, respectively.

9.11. Summary and outlook toward further improvements

Chapter 9 describes a search for long-lived charginos using disappearing tracks. The pure-higgsino scenario, in which a mass splitting between the lightest chargino and neutralinos is generated by radiative effects, is the focus of the analysis.

Two strategies for the determination of the background are shown. The first one uses different techniques to derive templates for each of the different background sources. For charged lepton scatters, a tag-and-probe technique is used to calculate a transfer factor. It captures the probability for a charged lepton to scatter and be reconstructed as a tracklet. The transfer factor is applied to charged leptons selected in control regions close to the signal region, with the addition of a momentum smearing to account for resolution differences. A single-track control region to which the electron transfer factor is applied is used as a shortcut

[10] The publication does not quote this number, it was determined from the expected limit data from the corresponding HEPData record.

for a hadron template. To determine fake background templates, tracklet quality criteria are inverted, to enrich fake contribution and suppress signal contamination. An attempt to determine the absolute normalization via a simultaneous fit to a low-$E_{\mathrm{T}}^{\mathrm{miss}}$ control region and the high-$E_{\mathrm{T}}^{\mathrm{miss}}$ signal region. Ultimately, due to low difference in template shape, the fit normalization is found to be unstable.

As an alternative approach, an inclusive ABCD method is used to determine a combined background estimate for all background sources, reusing the same low- and high-$E_{\mathrm{T}}^{\mathrm{miss}}$ regions, and the tracklet quality sidebands. The dominant systematic uncertainties affecting the signal and background are evaluated and taken into account. An uncertainty related to the ABCD method itself is also included explicitly. The modelling of the background by this method is found to be reasonable, and within uncertainties.

With this inclusive ABCD method, expected upper limits on the signal strength parameter for a pure-higgsino signal model are calculated and presented. A combination of real and pseudo-data is used to constrain the systematic uncertainties of the background prediction. Standalone limits using only three- or four-layer tracklets are presented, as is a combination of both, that is obtained by simultaneous fits. The combined limit is found to be an improvement over the individual limits. Using the upper signal strength limits, expected lower bounds on the allowed masses can be calculated. At the nominal luminosity of the used 2018 dataset, 58.5 fb^{-1}, this limit improves on the pure-higgsino limit most recently published by ATLAS at the time of writing. By scaling the dataset to the one used in the public result, it is verified that the four-layer-only limit from the publication can be reproduced. A tightened longitudinal impact parameter cut is found to substantially improve sensitivity of the three-layer selection, thereby also improving the combined sensitivity. By upscaling both cases to the full Run 2 luminosity, an expectation for an equivalent analysis using this dataset is derived. In the absence of a full Run 2 ATLAS public result, it is found to compare very favorably to the corresponding CMS publication.

With these results in mind, the following sections give some details on how the presented analysis can be improved further, using more advanced background estimation techniques, and an improved tracklet reconstruction algorithm.

9.11.1. Improved estimation of backgrounds

The main ATLAS publication uses an approach that is relatively similar to the template based approach presented in section 9.7. This gives confidence to the idea that this method is generally suitable for estimating the background, in spite of the difficulties encountered during the work for this thesis. If the limited shape differences can be overcome, a normalization can potentially be extracted from the fit.

Even more desirable than attempting to extract all background normalizations from a likelihood fit, would be an absolute prediction of the normalization of individual background sources. To this end, the charged lepton and hadron scatter templates could be used to obtain a sample reasonably pure in fake tracklets, by subtracting them. This fake tracklet sample could then be used to perform an absolute ABCD estimate of the fake background in

the signal region. In a similar fashion, the scatter templates could be used directly without attempting normalization, if the charged lepton and hadron control regions multiplied by the corresponding transfer factors yielded correct normalizations. This approach was attempted during the work for this thesis, but was found not to work due to the electron template strongly overpredicting the amount of scattered electrons. This breaks the subtraction method used for deriving a pure fake sample. Other ways of obtaining absolute normalization by means of different sidebands and control regions were studied, but ultimately no promising candidates could be identified.

Given an absolutely normalized estimate of the different background sources, the background modelling could likely be considerably improved, reducing uncertainties and finally increasing the sensitivity.

9.11.2. Reconstruction of soft secondary particles and vertex tagging

A major drawback of the reduction of the required number of Pixel hits in the tracklet reconstruction (see section 8.2) is the increase in the rate of random combinations being observed. The likelihood of a random combination of measurements passing the reconstruction and selection criteria rises in case of only three measurements being considered sufficient. On the one hand, this is simply because of the largely increased number of triplet combinations relative to the number of quadruplets. On the other hand, triplets are more likely to be reconstructed as a valid helix trajectory, since three points always lie on a circle, if the circle is chosen correctly. As this is not the case for four points, this trivial requirement already rejects a significant portion of random combinations of four hits.

In an attempt to find additional handles to suppress this contribution to the tracklet population, [217] proposes a new algorithm to reconstruct the soft $\pi^{\pm}$. While figure 8.1 shows the pion curling away entirely due to its low momentum, it can potentially reach the SCT detector, and deposit energy in its sensors. Depending on the signal model under study, the pion is likely to be rejected by the requirement of $p_{\mathrm{T}} > 500$ MeV of the standard track reconstruction, due to its transverse momentum. However, simply lowering this threshold would result in a significantly increased runtime of the standard track reconstruction, since a large amount of additional seed triplets would be fed into the subsequent algorithms.

To keep the runtime within reasonable limits, a region of interest is constructed around tracklet candidates, by all SCT hits within a cone of $\Delta R = \sqrt{\Delta\phi^2 + \Delta\theta^2} < 0.8$ along the direction from the beam spot to the last hit in the tracklet. This is shown in figure 9.38, for a $\tilde{\chi}_1^{\pm} \to \pi^{\pm}\tilde{\chi}_1^0$ decay, where the pion curls but reached the SCT. Here, the region of interest fully wraps all SCT hits that are produced by the pion. In ATLAS' solenoid field of 2 T, the curvature shown corresponds to a pion transverse momentum of about 150 MeV.

In another second-pass reconstruction stage, SCT-only track candidates are formed, and filtered for compatibility with the last hit of the tracklet. It is then used as a proxy for the expected $\tilde{\chi}_1^{\pm}$ decay vertex. This selection procedure operates under the assumption that the pion's measurements are reasonably collinear with the direction from the beam spot to the last tracklet hit, which was studied at the hit level in the public note. This behavior could

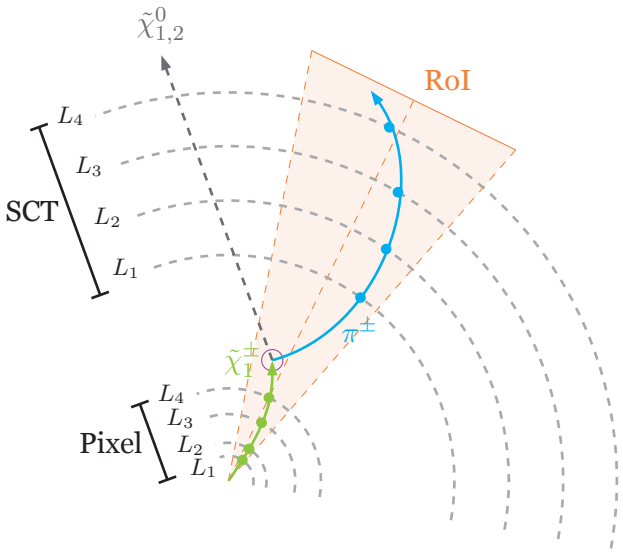


Figure 9.38.: Illustration of how a region of interest (RoI) is used to select SCT hits for soft-particle reconstruction. It shows a chargino decaying into a pion and a neutralino, producing a tracklet of Pixel hits, and a SCT track produced by the pion. The curvature of the pion track corresponds to about $p_\mathrm{T} \sim 150\,\mathrm{MeV}$.

not be reproduced between the $\tilde{\chi}^\pm_1$ and $\pi^\pm$ at truth level during sanity checks for this thesis. Here, a discrepancy in the ϕ direction is observed. It is believed to be a consequence of the way the simulation is set up to force $\tilde{\chi}^\pm_1$ truth particles to decay in a delayed fashion after propagation through the detector in `Geant4`. Another feature, which is also believed to be of related origin, is that the $\tilde{\chi}^\pm_1 \to \tilde{\chi}^0_1\pi^\pm$ vertex does not conserve four-momentum at truth-level. The efficiency to reconstruct a soft-pion from a chargino decay is evaluated in MC to be up to 75 %, depending on the $\pi^\pm$ momentum, and model parameters, and is found to be reasonably robust against pile-up.

Reconstructed soft-pion track candidates are then used to fit vertices with the tracklets that have been used as inputs for the procedure, where both types of tracks are filtered first to improve the fit quality. While a vertex reconstructed in this way is expected to have poor spatial resolution, due to only two tracks being used to build it, it can potentially be used to select tracklets which are likely to have decayed into a soft-pion. The efficiency of reconstructing a chargino-decay vertex in this manner, under the precondition that both particles were correctly reconstructed, is evaluated in [217] to be close to 90 %. A drop in efficiency is observed for low opening angles $\Delta R(\tilde{\chi}^\pm_1, \pi^\pm)$ between the chargino and the pion.

Both tagging tracklets with the existing of a pion via a secondary vertex, and placing requirements on the vertex properties were identified as potential candidates for improving an analysis using three-layer Pixel tracklets. The additional information could allow suppression of random combinations, which are expected to be dominant for the shorter-length tracklets. With these improved background rejection techniques, the signal over background ratio could be improved, thereby yielding additional sensitivity, especially at the lowest lifetimes where three-layer tracklets are expected to be most sensitive.

9.11.3. Future detectors and colliders

The LHC will undergo extensive upgrades until the end of the third long shutdown period. HL-LHC ([169], see chapter 5) is expected to deliver extensive additional datasets, due to a strongly increased instantaneous luminosity. ATLAS and the other LHC experiments in turn are conducting ambitious upgrade programs, to cope with these extreme conditions. ATLAS will replace its tracking system with the ITk [172–174], an all-silicon tracker designed from scratch for high pile-up.

Expected sensitivities for disappearing track searches were studied [242]. Lower expected mass limits for pure-wino (pure-higgsino) signals were found to be 850 GeV (250 GeV) at $\mathcal{L} = 3\,\text{ab}^{-1}$. The sensitivity at low lifetimes crucially depends on the radius of the innermost tracking layer. In order to have any hope of obtaining enough measurements to routinely reconstruct low lifetime charginos, this radius must be as low as possible. At the time of writing, a recent decision was made to construct an innermost ITk pixel layer at $r = 34\,\text{mm}$ (see section 5.3.1), with which sensitivity at lower lifetimes could potentially exceed these aforementioned predictions.

The future of high-energy colliders is unclear at the time of writing of this thesis. Different design studies are being pursued, with the Future Circular Collider (FCC) [243–245] being one of them. Both an electron-electron (FCC-ee) and proton-proton (FCC-hh) collider are being evaluated, the latter with a center-of-mass energy of up to 100 TeV. Alongside an energy increase, a raised instantaneous luminosity compared to HL-LHC is foreseen. Different studies [107, 246] have been conducted examining the potential sensitivity of disappearing track searches. Due to increased pile-up, shorter tracks are expected to suffer from increased fake background, reducing sensitivity. Depending on the final track reconstruction efficiency and background rejection, it is expected that FCC-hh will push lower mass bounds for both pure-wino and pure-higgsino beyond thermal upper limits, thereby having the potential to fully exclude them with less than $\mathcal{L} = 30\,\text{ab}^{-1}$.

10. Conclusion

Particle collisions and the analysis of their resulting particles are a critical tool for the research of particle physics. One aspect of particle detection is the reconstruction of particle tracks. Complex algorithms operate on measurements taken along particle trajectories by sensitive elements to recover properties of the particles. As algorithmic complexity scales with event activity, track reconstruction presents a significant challenge. Upcoming upgrades to the LHC and future colliders necessitate the improvement of track reconstruction software.

This thesis presented on two main areas of interest: development and improvement of particle track reconstruction software, in the form of major contributions to the ACTS toolkit, as well as a specific application of reconstructed tracks in the form of a search for a signal model with very short disappearing particle tracks.

Part II of this thesis gave an introduction to the ACTS project in general, and explained what the design goals and specific components are. It also showed how modern software concepts and best practices are leveraged to improve the quality and maintainability of the codebase. Subsequently, a series of specific contributions made to ACTS in the context of this thesis were described in detail.

The first sections introduced the model chosen and implemented to integrate ACTS back into the ATLAS tracking software. The work described here makes up the foundations of, and launched, the integration effort, that has since been joined by additional contributors. Specific focus is given to the description of the ATLAS Inner Detector geometry, which is a critical ingredient to most track reconstruction areas. Details were given on the interplay between ACTS-specific components and their counterparts in the ATLAS software, complemented by descriptions of both. Lastly, thorough tests and checks were presented that validate the correctness and viability of the implementations.

As ACTS is supposed to be a suitable for the ATLAS software even after the completion of the Inner Tracker (ITk) upgrade, a feasibility study to determine if that geometry can be accommodated was conducted. No issues were observed. As the ITk uses a new type of endcap strip sensor, existing software only modelled them in an approximate fashion. A review of this new type of sensor shape was given, contrasting it with the existing SCT sensors. It then described a modelling of the new sensor shape that was developed for this thesis, which is able to better capture the geometric sensor structure. The implementation was tested for viability and compatibility and found to provide both.

In the interest of providing a complete geometry description of all ATLAS components, the calorimeter geometry needs to be integrated. A novel approach that attempts to overcome the fragility and fine hand tuning of the existing approach chosen in ATLAS was shown.

Leveraging concepts from computer graphics, it uses ray tracing and frustum intersections in combination with bounding volume hierarchies to provide a fast and generic navigation query infrastructure. The application of this new navigation model to the ATLAS calorimeter readout geometry was evaluated and shown to work as expected. Performance characterizations completed the discussion of the navigation model.

ACTS uses compile-time programming to ensure high runtime performance. One particular area where this is the case is the Event Data Model. Several improvements to the EDM were introduced, that were developed in the context of this thesis. One is a reimplementation of a specific compile-time optimization in a more maintainable fashion, carefully matching previous performance. Another one is the new implementation of a data structure usable for filtering applications. This data structure was designed with the Kalman Filter and Combinatorial Kalman Filter use case in mind. The Kalman Filter was refactored as part of the work to stress test and validate the new data structure.

Handling of time-dependent parameters during data-taking is a critical aspect of event reconstruction. In the tracking domain, time-dependent misalignment plays an important role in this context. As part of the work for this thesis, a concurrent misalignment handling strategy was developed, working in tandem with the multi-threaded infrastructure provided by the ATLAS event-processing framework. Specific adaptations to ACTS were made to facilitate this integration. Showcases of the implementation of required components needed to connect the ACTS geometry with this infrastructure were given. To this end, additional algorithms were added that translate available misalignment data into a format usable by ACTS, and setting up the code to make full use of this data. Correct propagation of the information was asserted, and found to compare favorably with alternative approaches.

Work on ACTS is far from complete. The work presented in this thesis can and will be continued into the future. A rewrite of the navigation component will enable full usage of the new navigation model shown. Improvements to the EDM will further reduce compile-time impact, and provide more seamless connectivity to experiments. ATLAS integration efforts are ramping up, and additional ACTS components will be prepared on the road to Run 3. Modelling of the ATLAS Run 4 upgrade geometry will progress, likely using a polar coordinate system based description.

Part III of this thesis is focussed on an analysis whose goal is a search for a specific SUSY model that is expected to result in very short particle tracks. After travelling through the innermost part of the ATLAS tracker, the Pixel detector, the tracks disappear, as the underlying particle decays. This thesis reports on incorporating dedicated reconstruction techniques, that were improved upon to recover even shorter tracks with as little as three hits. With these tracklets, the achievable sensitivity is quantified. Proton-proton data taken at $\sqrt{s} = 13\,\text{TeV}$ in 2018 is used for this purpose, for which the new reconstruction was available. Two different data-driven background estimation approaches are evaluated, the latter of which is found to be useable. With this background estimation in addition to a simulation-based signal expectation, systematic uncertainties involved in the analysis method are determined and discussed. Finally, a statistical analysis is conducted, and provides results in the form of

expected upper limits on the signal strength of the signal model in question, as well as expected lower mass limits. Comparisons to both existing ATLAS and CMS results are made. Expected limits found in this thesis compares favorably in performance to these results.

The very short three-layer tracklets were found to be a viable improvement of the search strategy, since they are expected be able to improve sensitivity at a given level of statistics. By incorporating pion tagging as an additional background rejection technique, the impact of three-layer tracklets could be improved upon further in the future. As future instances of this kind of analysis will have to be adapted for the upgraded ATLAS tracker, or a completely new detector as would be the case for FCC-hh, three-layer tracklets can serve as a strong optimization of the sensitivity achievable with the ID geometry.

Appendix

A. ACTS development

```
template <typename value_t, size_t DIM, size_t SIDES>
template <size_t D, std::enable_if_t<D == 3, int>>
Acts::Frustum<value_t, DIM, SIDES>::Frustum(const VertexType& origin,
                                            const VertexType& dir,
                                            value_type opening_angle)
    : m_origin(origin) {
  static_assert(SIDES > 2, "3D frustum must have 3 or more sides");
  assert(opening_angle < M_PI);
  using angle_axis_t = Eigen::AngleAxis<value_type>;

  const VertexType ldir = VertexType::UnitZ();
  const VertexType lup = VertexType::UnitX();

  transform_type transform;
  transform = (Eigen::Quaternion<value_type>().setFromTwoVectors(ldir, dir));

  m_normals[0] = ldir;

  const value_type phi_sep = 2 * M_PI / sides;
  transform_type rot;
  rot = angle_axis_t(phi_sep, ldir);

  value_type half_opening_angle = opening_angle / 2.;
  auto calculate_normal =
      [&ldir, &half_opening_angle](const VertexType& out) -> VertexType {
    const VertexType tilt_axis = -1 * out.cross(ldir);
    return (-1 * (angle_axis_t(half_opening_angle, tilt_axis) * out))
        .normalized();
  };

  VertexType current_outward = lup;
  m_normals[1] = calculate_normal(current_outward);

  for (size_t i = 1; i < sides; i++) {
    current_outward = rot * current_outward;
    m_normals[i + 1] = calculate_normal(current_outward);
  }

  for (auto& normal : m_normals) {
    normal = transform * normal;
  }
}
```

Listing A.1.: Core/include/Acts/Utilities/Frustum.ipp

```cpp
template <typename entity_t, typename value_t, size_t DIM>
template <size_t sides>
bool Acts::AxisAlignedBoundingBox<entity_t, value_t, DIM>::intersect(
    const Frustum<value_type, DIM, sides>& fr) const {
  const auto& normals = fr.normals();
  // Transform vmin and vmax into the coordinate system, at which the frustum is
  // located at the coordinate origin.
  const vertex_array_type fr_vmin = m_vmin - fr.origin();
  const vertex_array_type fr_vmax = m_vmax - fr.origin();

  // For each plane, find the p-vertex, which is the vertex that is at the
  // furthest distance from the plane *along* its normal direction.
  // See Fig. 2 in [2].
  VertexType p_vtx;
  // for loop, we could eliminate this, probably,
  // but sides+1 is known at compile time, so the compiler
  // will most likely unroll the loop
  for (size_t i = 0; i < sides + 1; i++) {
    const VertexType& normal = normals[i];

    // for AABBs, take the component from the min vertex, if the normal
    // component is negative, else take the component from the max vertex.
    p_vtx = (normal.array() < 0).template cast<value_type>() * fr_vmin +
            (normal.array() >= 0).template cast<value_type>() * fr_vmax;

    // Check if the p-vertex is at positive or negative direction along the
    // If the p vertex is along negative normal direction *once*, the box is
    // outside the frustum, and we can terminate early.
    if (p_vtx.dot(normal) < 0) {
      // p vertex is outside on this plane, box must be outside
      return false;
    }
  }

  // If we get here, no p-vertex was outside, so box intersects or is
  // contained. We don't care, so report 'intersect'
  return true;
}
```

Listing A.2.

option	purpose
ACTS_PARAMETER_DEFINITIONS_HEADER	Use a different (track) parameter definitions header
ACTS_BUILD_DD4HEP_PLUGIN	Build DD4HEP plugin
ACTS_BUILD_DIGITIZATION_PLUGIN	Build Digitization plugin
ACTS_BUILD_IDENTIFICATION_PLUGIN	Build Identification plugin
ACTS_BUILD_JSON_PLUGIN	Build Json plugin
ACTS_USE_BUNDLED_NLOHMANN_JSON	Use external `nlohmann::json`. If not enabled, a copy of the library will be downloaded and built automatically.
ACTS_BUILD_TGEO_PLUGIN	Build TGeo plugin
ACTS_BUILD_LEGACY	Build legacy plugin
ACTS_BUILD_FATRAS	Build FAst TRAcking Simulation package
ACTS_BUILD_EXAMPLES	Build standalone examples
ACTS_BUILD_EXAMPLES_DD4HEP	Build DD4hep-based [188] code in the examples
ACTS_BUILD_EXAMPLES_GEANT4	Build Geant4-based [132] code in the examples
ACTS_BUILD_EXAMPLES_HEPMC3	Build HepMC3-based [247] code in the examples
ACTS_BUILD_EXAMPLES_PYTHIA8	Build Pythia8-based [130] code in the examples
ACTS_BUILD_BENCHMARKS	Build benchmarks
ACTS_BUILD_UNITTESTS	Build unit tests
ACTS_BUILD_INTEGRATIONTESTS	Build integration tests
ACTS_BUILD_EVERYTHING	Build with most options enabled (except documentation
ACTS_BUILD_DOC	Build documentation

Table A.1.: Overview of all available options in the `CMake` configuration and their purpose at the time of writing.

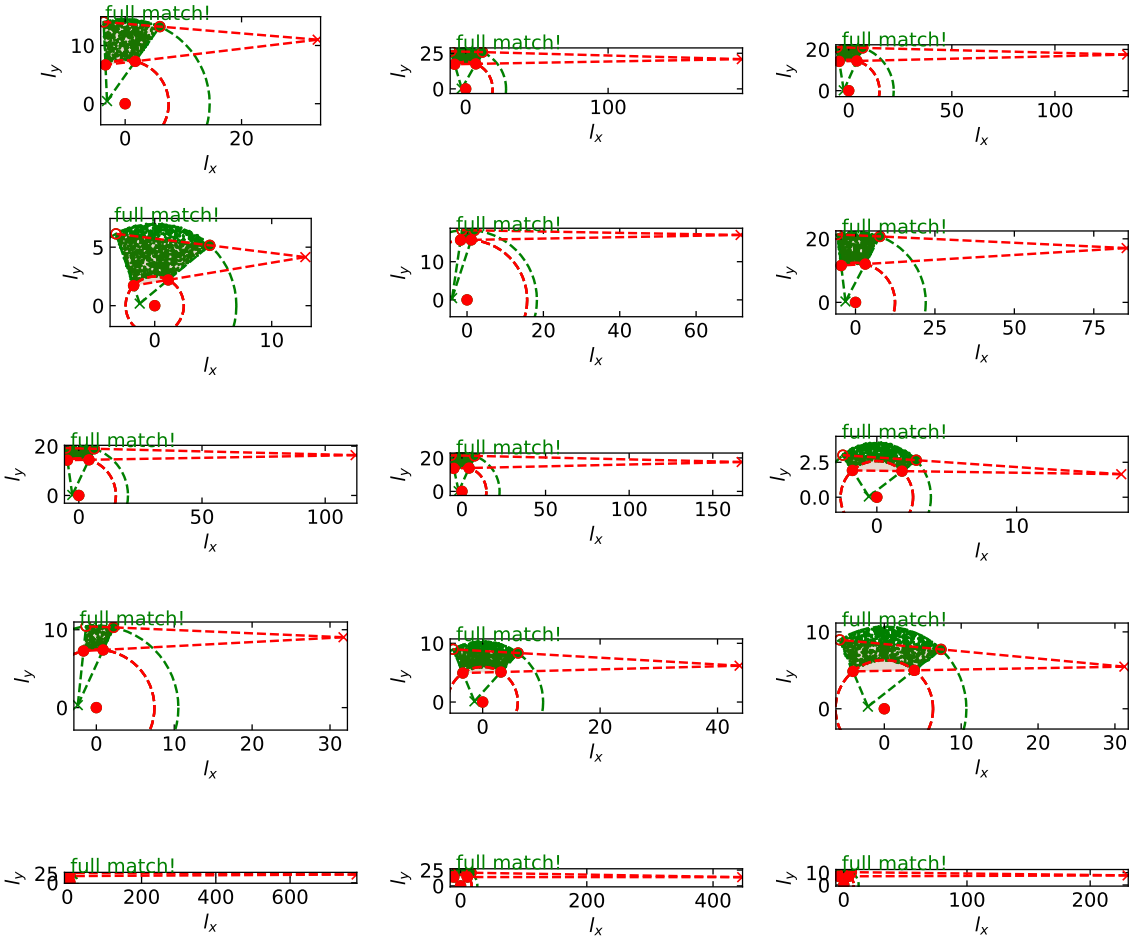


Figure A.1.

B. Disappearing track search

dsid	AMI tag	channel	m [GeV]	# events	eff. σ[pb]
448451	e7685_e5984_s3475_r11620_r11567_p4029	C1C1	160	50000	0.1350
448449	e7685_e5984_s3475_r11620_r11567_p4029	C1pN1	160	25000	0.1734
448450	e7685_e5984_s3475_r11620_r11567_p4029	C1mN1	160	25000	0.0960
449825	e7685_e5984_s3475_r11620_r11567_p4029	C1C1	200	50000	0.0669
449823	e7685_e5984_s3475_r11620_r11567_p4029	C1pN1	200	25000	0.0861
449824	e7685_e5984_s3475_r11620_r11567_p4029	C1mN1	200	25000	0.0453
448454	e7685_e5984_s3475_r11620_r11567_p4029	C1C1	240	50000	0.0371
448452	e7685_e5984_s3475_r11620_r11567_p4029	C1pN1	240	25000	0.0468
448453	e7685_e5984_s3475_r11620_r11567_p4029	C1mN1	240	25000	0.0238
448448	e7157_s3353_r10724_p3759	C1C1	120	10000	0.3178
448446	e7157_s3353_r10724_p3759	C1pN1	120	10000	0.4176
448447	e7157_s3353_r10724_p3759	C1mN1	120	10000	0.2432
448451	e7157_s3353_r10724_p3759	C1C1	160	10000	0.1348
448449	e7157_s3353_r10724_p3759	C1pN1	160	10000	0.1755
448450	e7157_s3353_r10724_p3759	C1mN1	160	10000	0.0963
448454	e7157_s3353_r10724_p3759	C1C1	240	10000	0.0369
448452	e7157_s3353_r10724_p3759	C1pN1	240	10000	0.0472
448453	e7157_s3353_r10724_p3759	C1mN1	240	10000	0.0238

Table B.1.: Monte Carlo samples used for the signal processes. Samples ending in *p4029* include the three-layer tracklet reconstruction and SCT hit-filter, while the samples ending in *p3759* do not.

		$e^{\pm}$			$\mu^{\pm}$		
$p_{\mathrm{T}}^{\mathrm{low}}$	$p_{\mathrm{T}}^{\mathrm{high}}$	z	σ	α	z	σ	α
25	35	−0.2033	17.0057	1.8586	−0.2485	14.8360	1.7168
35	45	−0.1461	15.4248	1.8224	−0.1890	14.2126	1.6638
45	60	−0.1268	14.4909	1.6557	−0.1339	13.6387	1.6187
60	100	−0.2056	13.9007	1.5442	−0.2804	13.4409	1.6824
100	200	−0.2139	14.0300	1.6374	0.0019	13.2081	1.6442

Table B.2.: Parameters extracted from a double-sided crystal-ball function fit to the q/p_{T} resolution distribution between standard muon tracks, and a tracklet refit using only their Pixel hits.

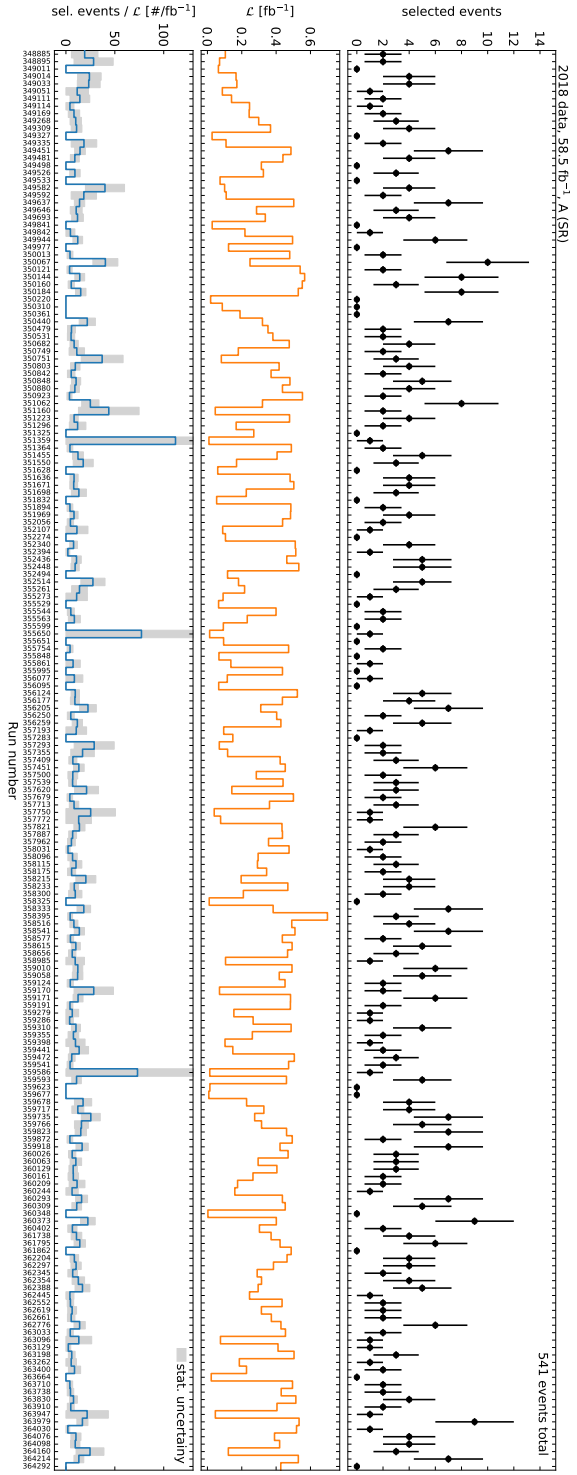


Figure B.1.: Number of selected events as a function of run number. Also shown is the integrated luminosity for each run, as well as the effective number selected events divided by the integrated luminosity.

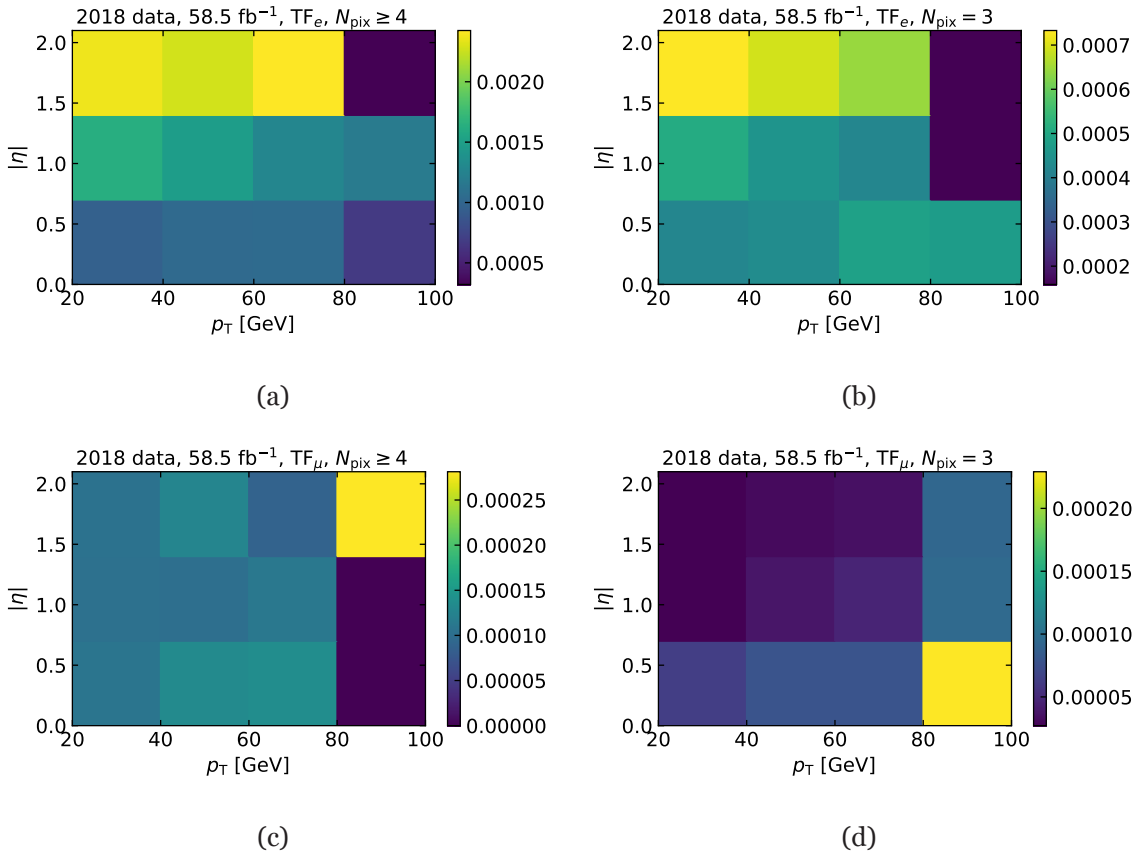


Figure B.2.: Transfer factors for electrons and muons as a function of p_{T} and η. Both three- and four-layer tracklets are shown.

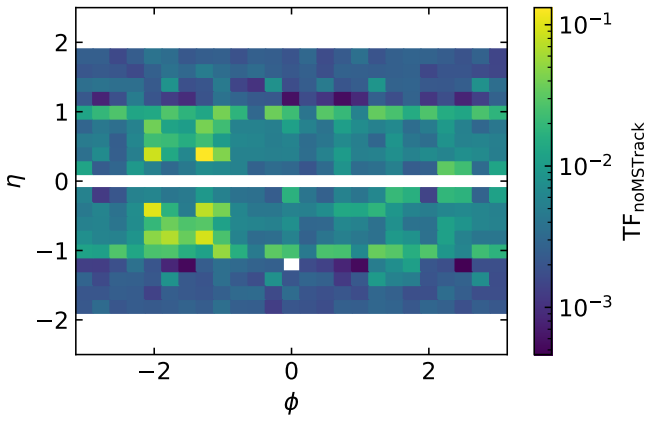
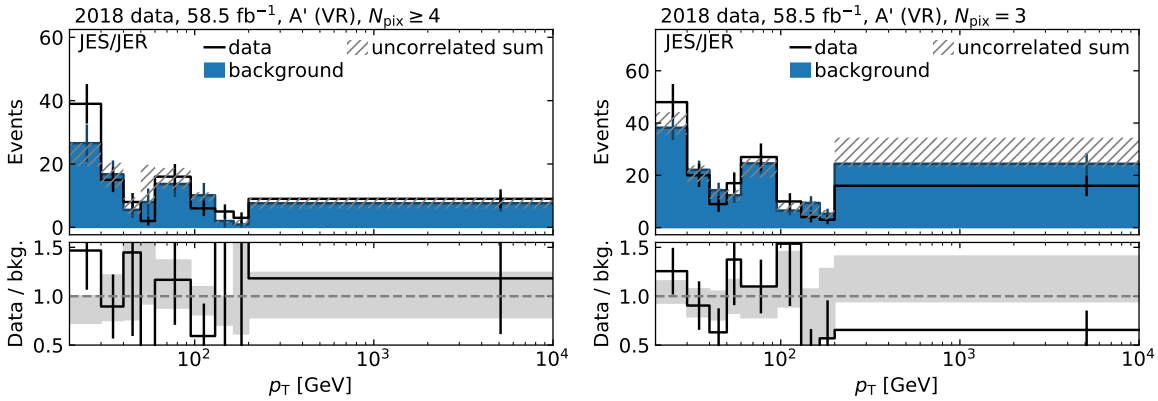


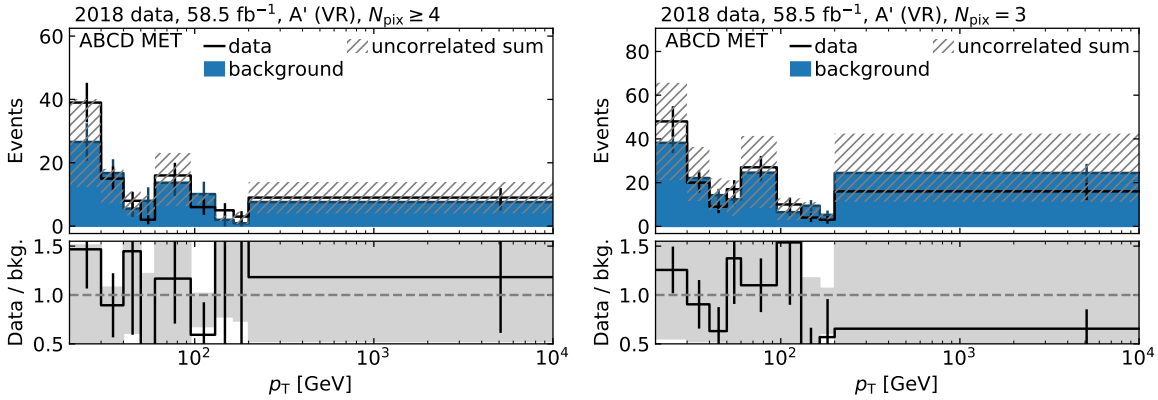
Figure B.3.: Transfer factor corresponding to the probability for a muon to be reconstructed without an MS-track, as a function of muon p_{T} and η. The values vary from about 10^{-1} at about $|\eta| \sim 1$, the transition region between barrel and endcap, and at $\phi \sim -\frac{\pi}{2}$ where the feet of the detector are located. Values as low as 10^{-3} are found elsewhere. Data taken from [237].

B.1. Systematic uncertainties



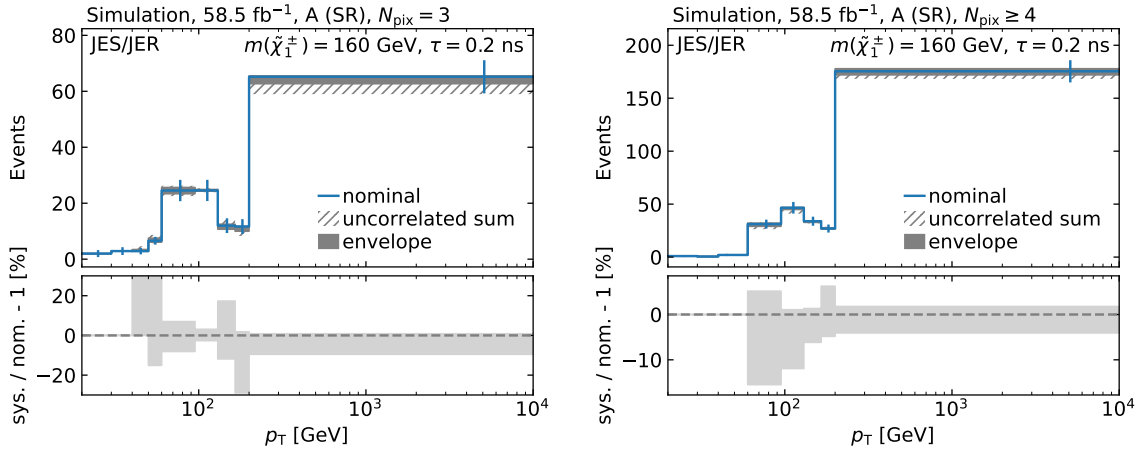
(a) (b)

Figure B.4.: Transverse momentum data distributions in the validation region for three- and four-layer tracklets. Also shown is the nominal background estimate, as well as statistical and systematic uncertainties related to the ABCD E_T^{miss} thresholds. The ratio between data and background, as well as nominal and varied background are given in the lower panel.



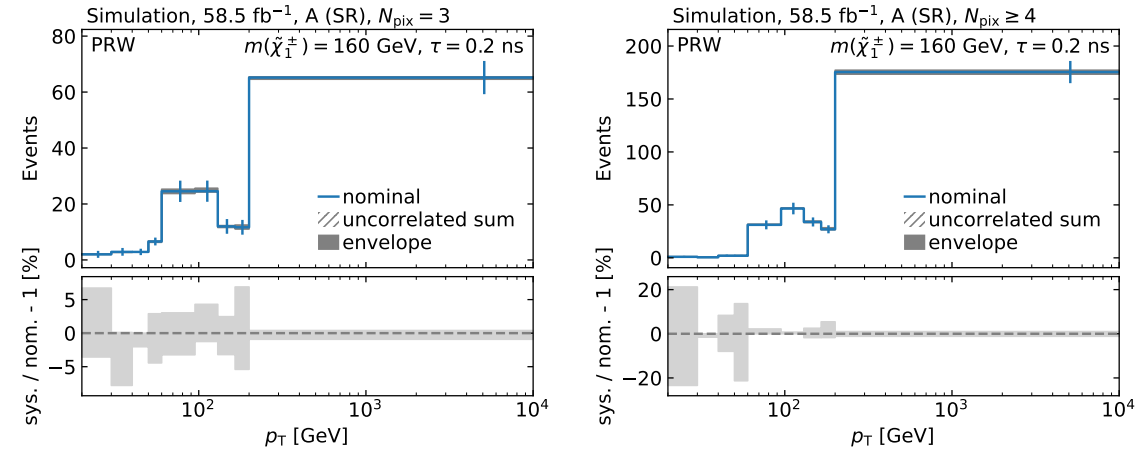
(a) (b)

Figure B.5.: Transverse momentum data distributions in the signal region for three- and four-layer tracklets. Also shown is the nominal background estimate, as well as statistical and systematic uncertainties related to the jet energy scale. The ratio between data and background, as well as nominal and varied background are given in the lower panel.



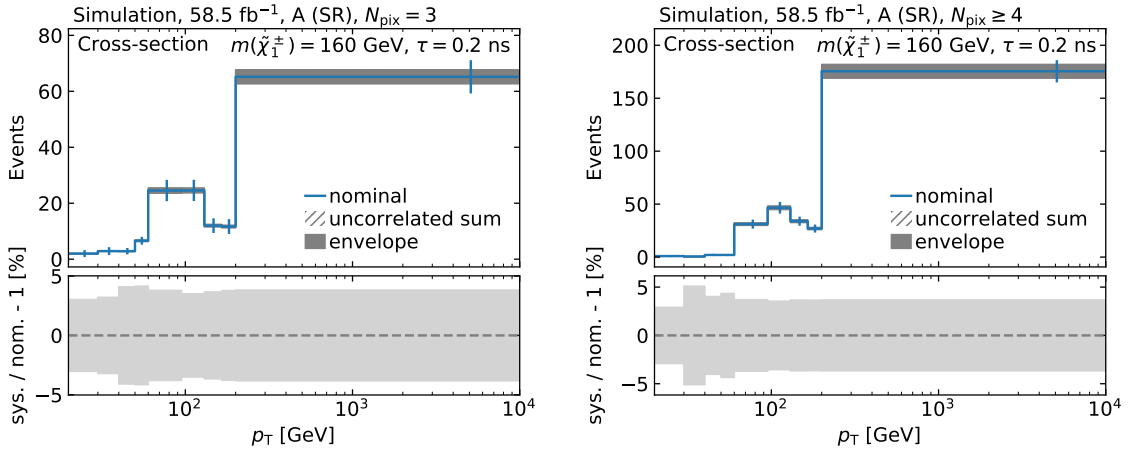
(a) (b)

Figure B.6.: Transverse momentum signal MC distributions in the signal region for three- and four-layer tracklets. Shown is the nominal signal prediction at $m(\tilde{\chi}^{\pm}) = 160$ GeV and $\tau = 0.2$ ns, as well as statistical and JESrelated systematic uncertainties. The ratio between nominal and varied background signal predictions are given in the lower panel.



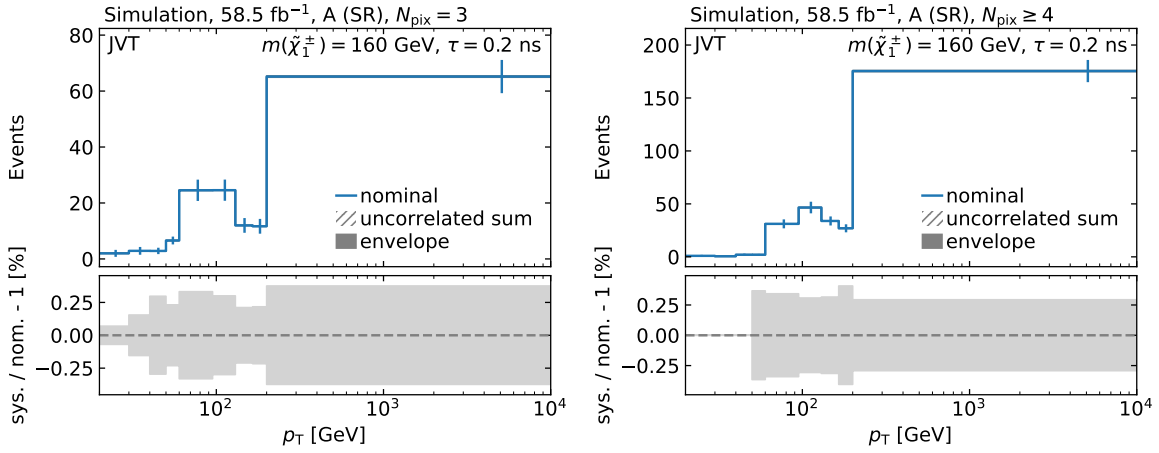
(a) (b)

Figure B.7.: Transverse momentum signal MC distributions in the signal region for three- and four-layer tracklets. Shown is the nominal signal prediction at $m(\tilde{\chi}^{\pm}) = 160$ GeV and $\tau = 0.2$ ns, as well as statistical and Pileup-modellingrelated systematic uncertainties. The ratio between nominal and varied background signal predictions are given in the lower panel.



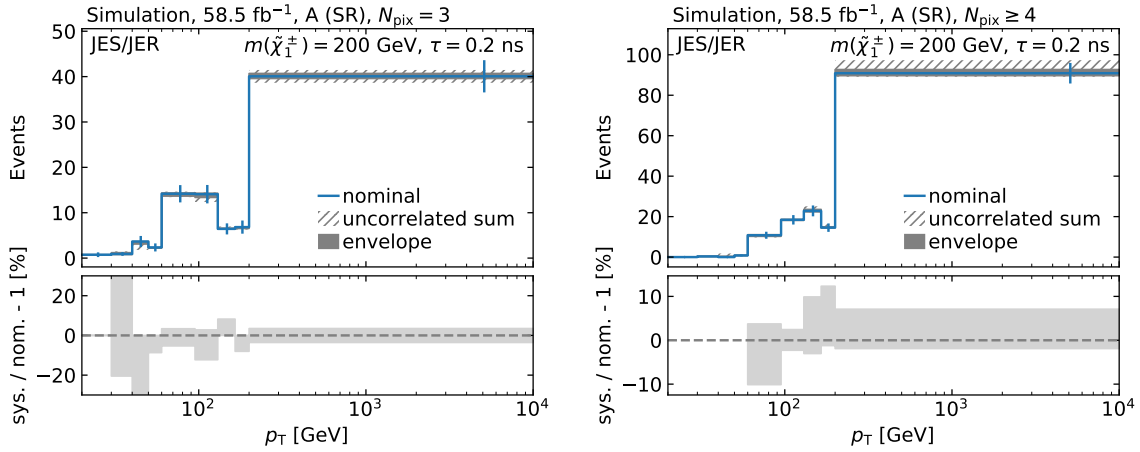
(a) (b)

Figure B.8.: Transverse momentum signal MC distributions in the signal region for three- and four-layer tracklets. Shown is the nominal signal prediction at $m(\tilde{\chi}^{\pm}) = 160$ GeV and $\tau = 0.2$ ns, as well as statistical and Cross-sectionrelated systematic uncertainties. The ratio between nominal and varied background signal predictions are given in the lower panel.



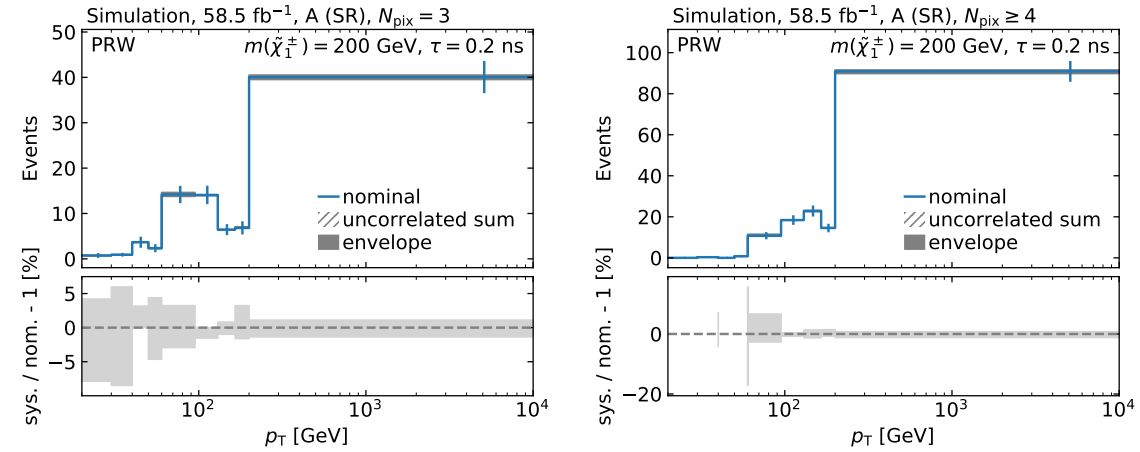
(a) (b)

Figure B.9.: Transverse momentum signal MC distributions in the signal region for three- and four-layer tracklets. Shown is the nominal signal prediction at $m(\tilde{\chi}^{\pm}) = 160$ GeV and $\tau = 0.2$ ns, as well as statistical and JVTrelated systematic uncertainties. The ratio between nominal and varied background signal predictions are given in the lower panel.



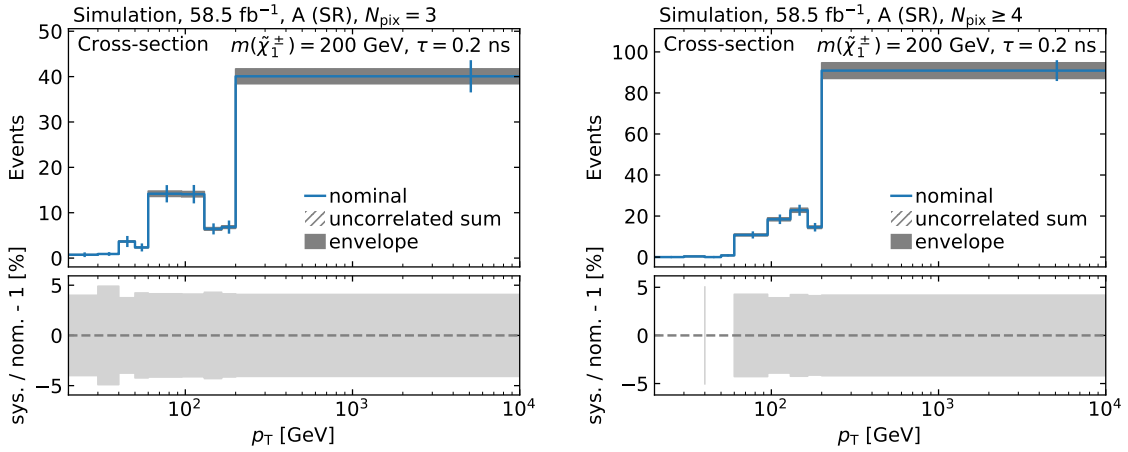
(a) (b)

Figure B.10.: Transverse momentum signal MC distributions in the signal region for three- and four-layer tracklets. Shown is the nominal signal prediction at $m(\tilde{\chi}^{\pm}) = 200$ GeV and $\tau = 0.2$ ns, as well as statistical and JESrelated systematic uncertainties. The ratio between nominal and varied background signal predictions are given in the lower panel.



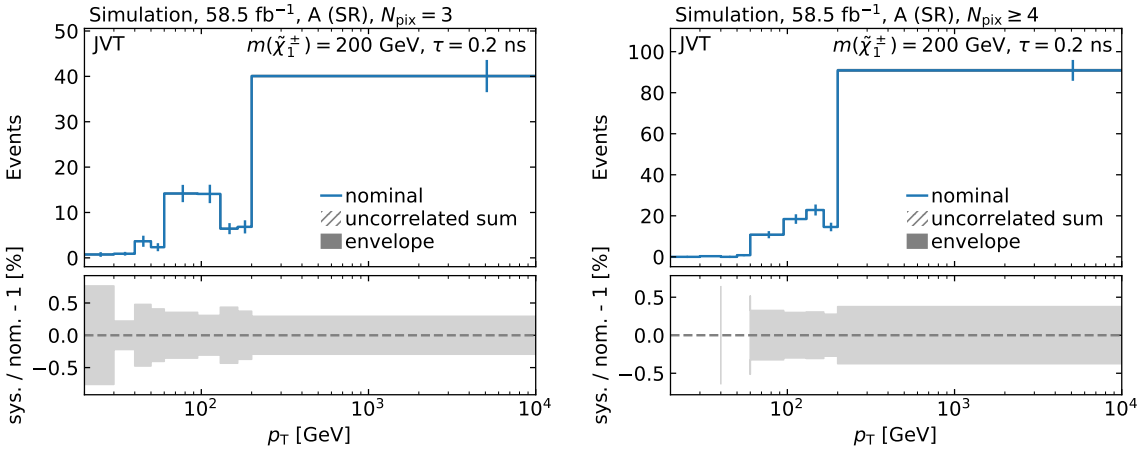
(a) (b)

Figure B.11.: Transverse momentum signal MC distributions in the signal region for three- and four-layer tracklets. Shown is the nominal signal prediction at $m(\tilde{\chi}^{\pm}) = 200$ GeV and $\tau = 0.2$ ns, as well as statistical and Pileup-modellingrelated systematic uncertainties. The ratio between nominal and varied background signal predictions are given in the lower panel.



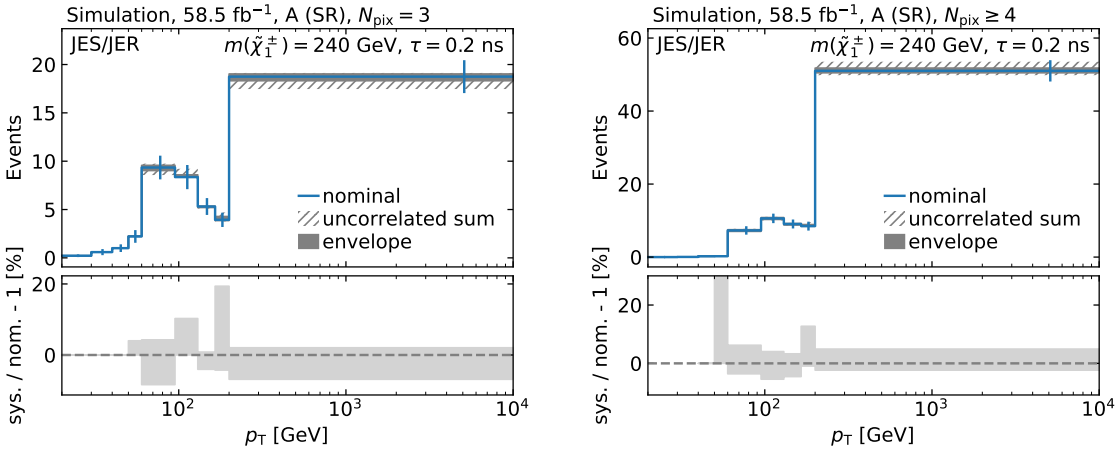
(a) (b)

Figure B.12.: Transverse momentum signal MC distributions in the signal region for three- and four-layer tracklets. Shown is the nominal signal prediction at $m(\tilde{\chi}^{\pm}) = 200$ GeV and $\tau = 0.2$ ns, as well as statistical and Cross-sectionrelated systematic uncertainties. The ratio between nominal and varied background signal predictions are given in the lower panel.



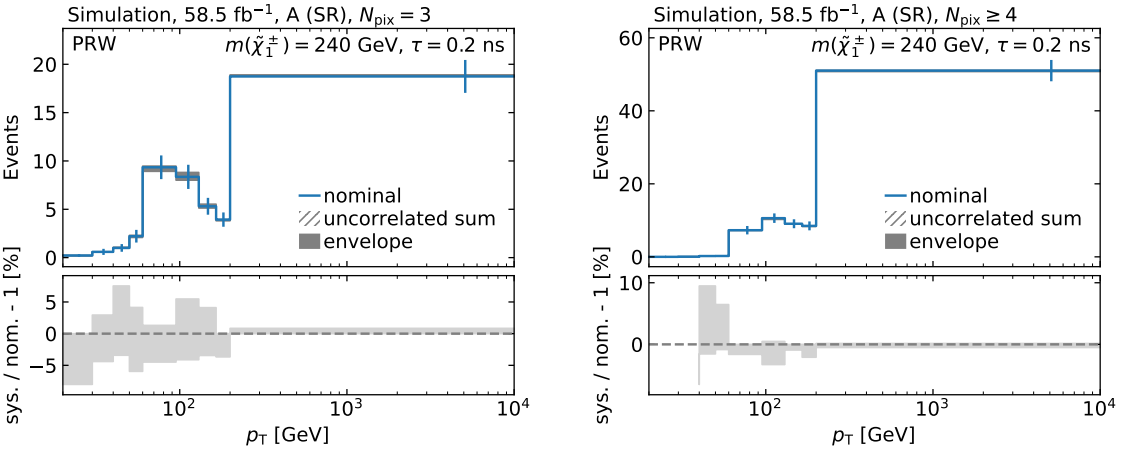
(a) (b)

Figure B.13.: Transverse momentum signal MC distributions in the signal region for three- and four-layer tracklets. Shown is the nominal signal prediction at $m(\tilde{\chi}^{\pm}) = 200$ GeV and $\tau = 0.2$ ns, as well as statistical and JVTrelated systematic uncertainties. The ratio between nominal and varied background signal predictions are given in the lower panel.



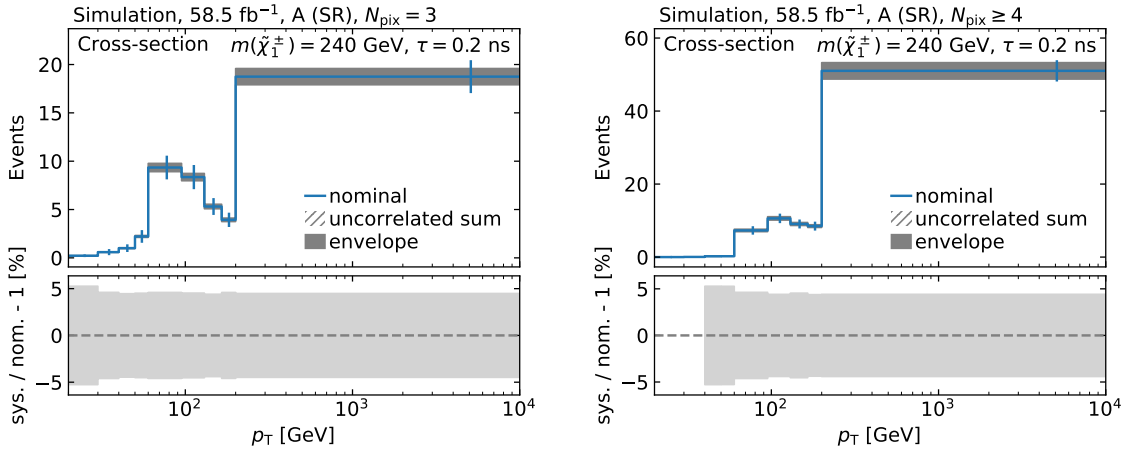
(a) (b)

Figure B.14.: Transverse momentum signal MC distributions in the signal region for three- and four-layer tracklets. Shown is the nominal signal prediction at $m(\tilde{\chi}^{\pm}) = 240$ GeV and $\tau = 0.2$ ns, as well as statistical and JESrelated systematic uncertainties. The ratio between nominal and varied background signal predictions are given in the lower panel.



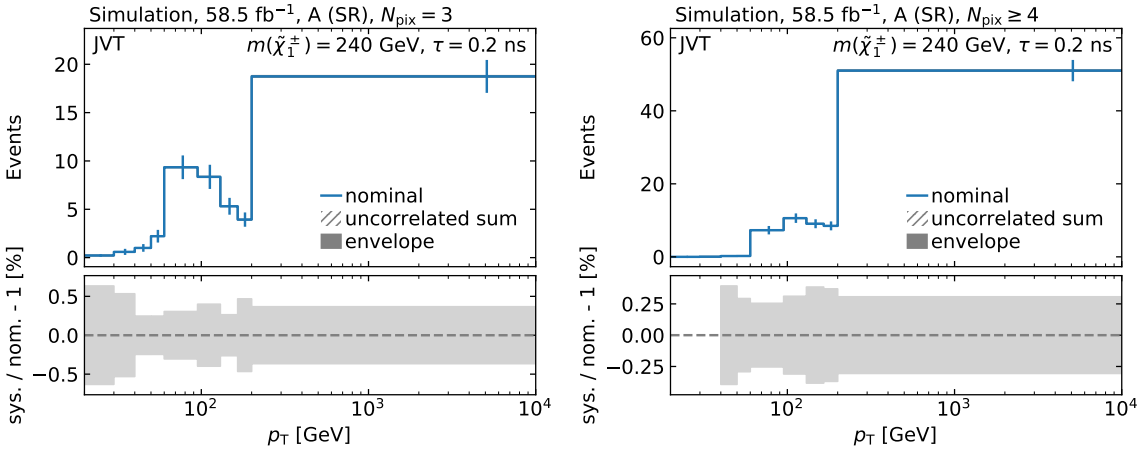
(a) (b)

Figure B.15.: Transverse momentum signal MC distributions in the signal region for three- and four-layer tracklets. Shown is the nominal signal prediction at $m(\tilde{\chi}^{\pm}) = 240$ GeV and $\tau = 0.2$ ns, as well as statistical and Pileup-modellingrelated systematic uncertainties. The ratio between nominal and varied background signal predictions are given in the lower panel.



(a) (b)

Figure B.16.: Transverse momentum signal MC distributions in the signal region for three- and four-layer tracklets. Shown is the nominal signal prediction at $m(\tilde{\chi}^{\pm}) = 240$ GeV and $\tau = 0.2$ ns, as well as statistical and Cross-sectionrelated systematic uncertainties. The ratio between nominal and varied background signal predictions are given in the lower panel.



(a) (b)

Figure B.17.: Transverse momentum signal MC distributions in the signal region for three- and four-layer tracklets. Shown is the nominal signal prediction at $m(\tilde{\chi}^{\pm}) = 240$ GeV and $\tau = 0.2$ ns, as well as statistical and JVTrelated systematic uncertainties. The ratio between nominal and varied background signal predictions are given in the lower panel.

B.2. Pre-fit and post-fit distributions

B.2.1. Individual fits

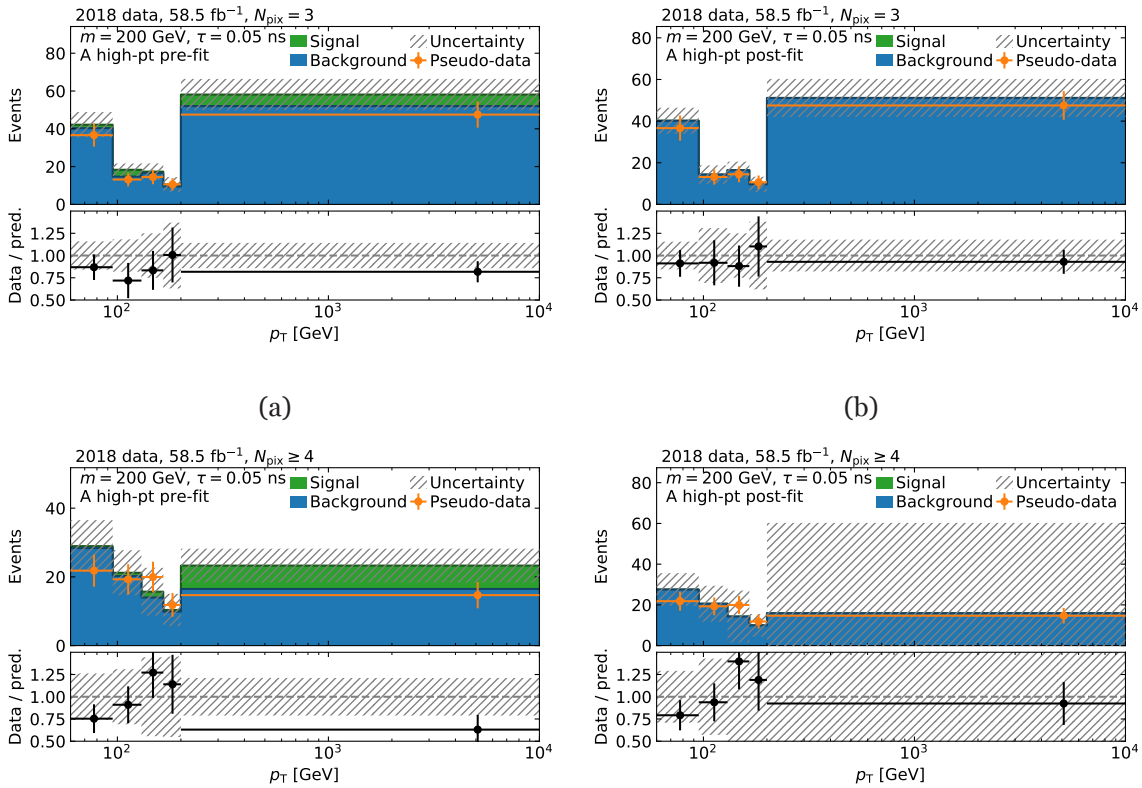


Figure B.18.: Transverse momentum distributions pre- and post-fit in the A_H region. The fit is performed for three- and four-layer distributions separately. Shown are (pseudo-)data, background estimate and a signal hypothesis for $\tau(\tilde{\chi}_1^{\pm}) = 0.05\,\text{ns}$ and $m = 200\,\text{GeV}$. Statistical and systematic uncertainties are displayed. The lower panel shows the ratio of (pseudo-)data and signal plus background prediction, as well as relative uncertainties.

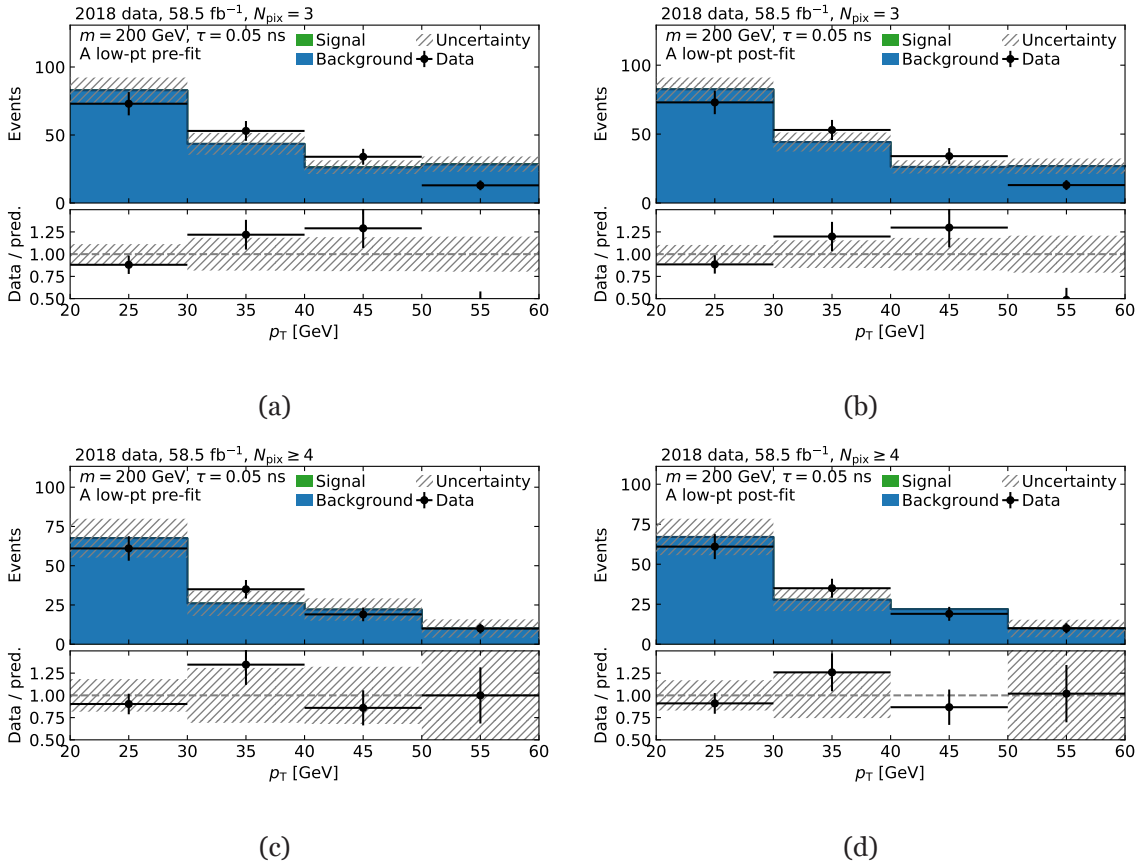


Figure B.19.: Transverse momentum distributions pre- and post-fit in the A_L region. The fit is performed for three- and four-layer distributions separately. Shown are (pseudo-)data, background estimate and a signal hypothesis for $\tau(\tilde{\chi}_1^{\pm}) = 0.05\,\text{ns}$ and $m = 200\,\text{GeV}$. Statistical and systematic uncertainties are displayed. The lower panel shows the ratio of (pseudo-)data and signal plus background prediction, as well as relative uncertainties.

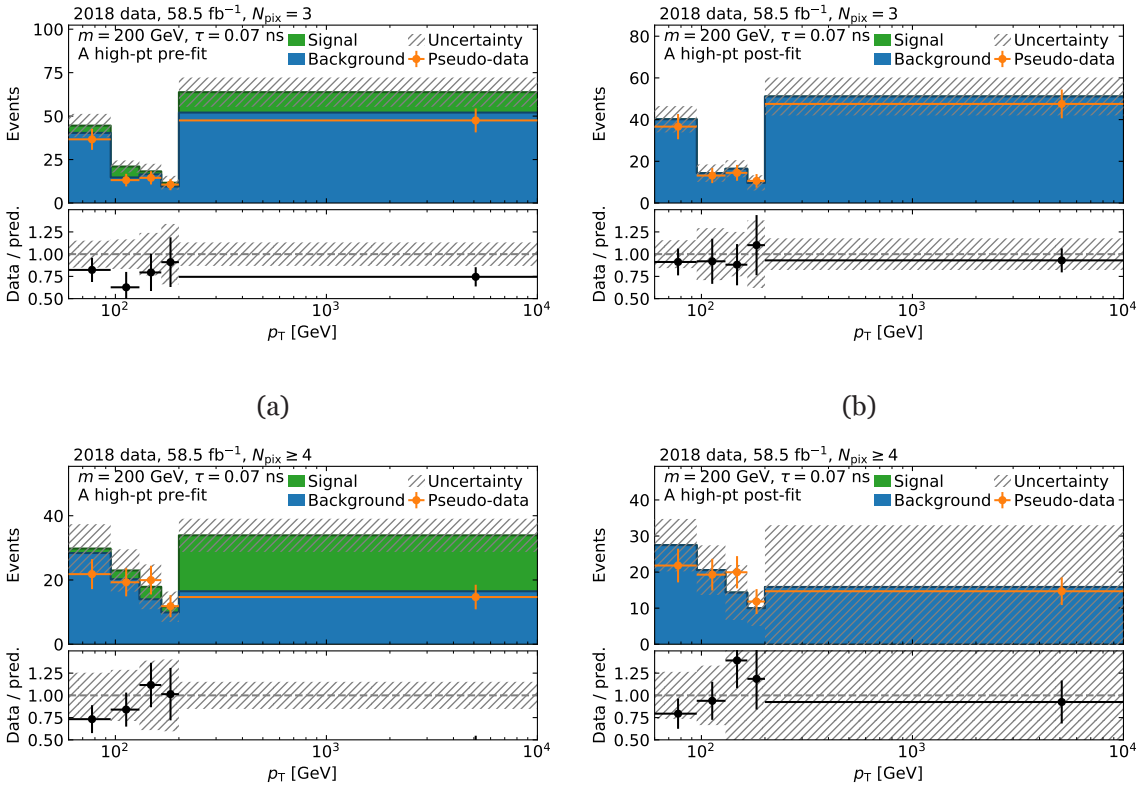


Figure B.20.: Transverse momentum distributions pre- and post-fit in the A_H region. The fit is performed for three- and four-layer distributions separately. Shown are (pseudo-)data, background estimate and a signal hypothesis for $\tau(\tilde{\chi}_1^{\pm}) = 0.07\,\text{ns}$ and $m = 200\,\text{GeV}$. Statistical and systematic uncertainties are displayed. The lower panel shows the ratio of (pseudo-)data and signal plus background prediction, as well as relative uncertainties.

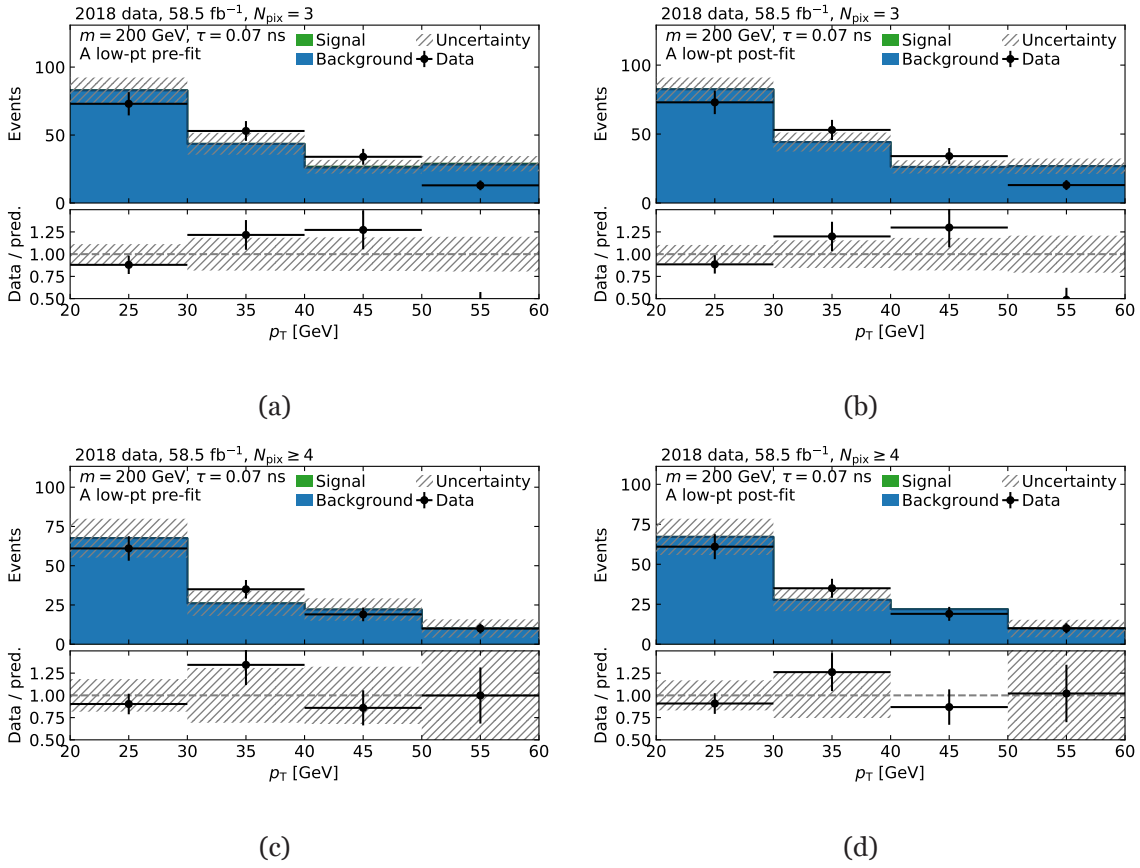


Figure B.21.: Transverse momentum distributions pre- and post-fit in the A_L region. The fit is performed for three- and four-layer distributions separately. Shown are (pseudo-)data, background estimate and a signal hypothesis for $\tau(\tilde{\chi}_1^{\pm}) = 0.07\,\text{ns}$ and $m = 200\,\text{GeV}$. Statistical and systematic uncertainties are displayed. The lower panel shows the ratio of (pseudo-)data and signal plus background prediction, as well as relative uncertainties.

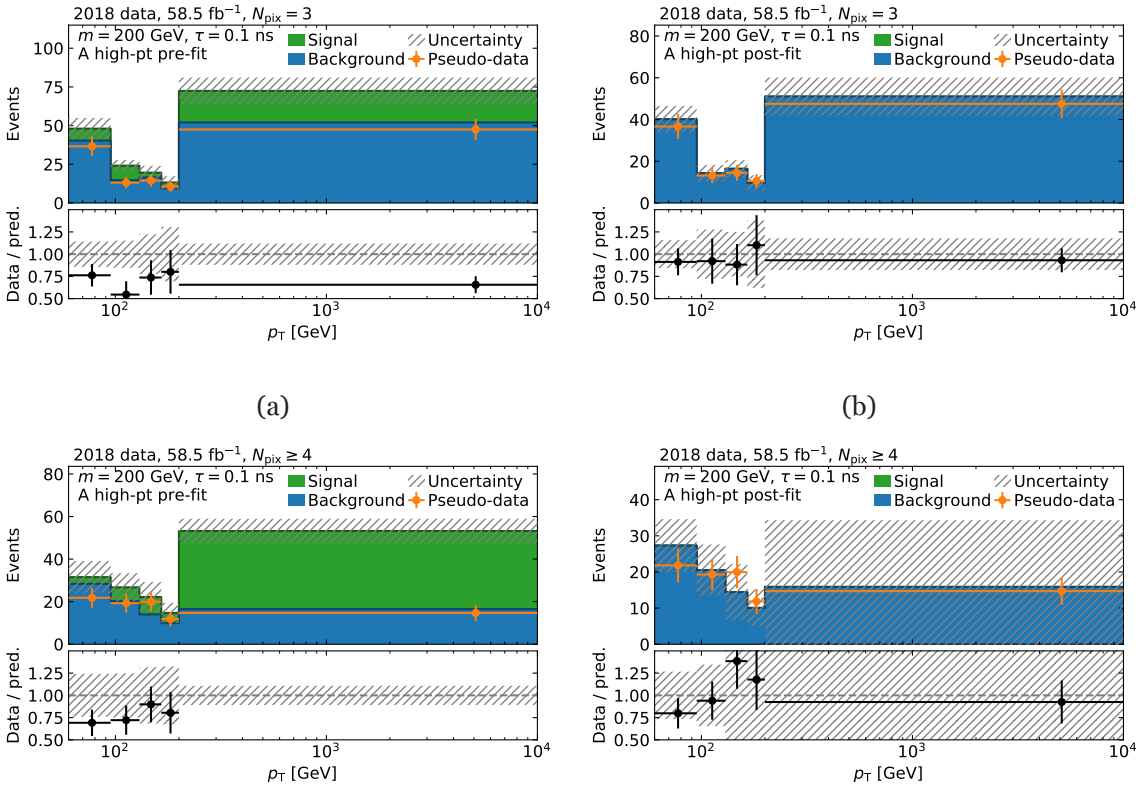


Figure B.22.: Transverse momentum distributions pre- and post-fit in the A_H region. The fit is performed for three- and four-layer distributions separately. Shown are (pseudo-)data, background estimate and a signal hypothesis for $\tau(\tilde{\chi}_1^{\pm}) = 0.1\,\text{ns}$ and $m = 200\,\text{GeV}$. Statistical and systematic uncertainties are displayed. The lower panel shows the ratio of (pseudo-)data and signal plus background prediction, as well as relative uncertainties.

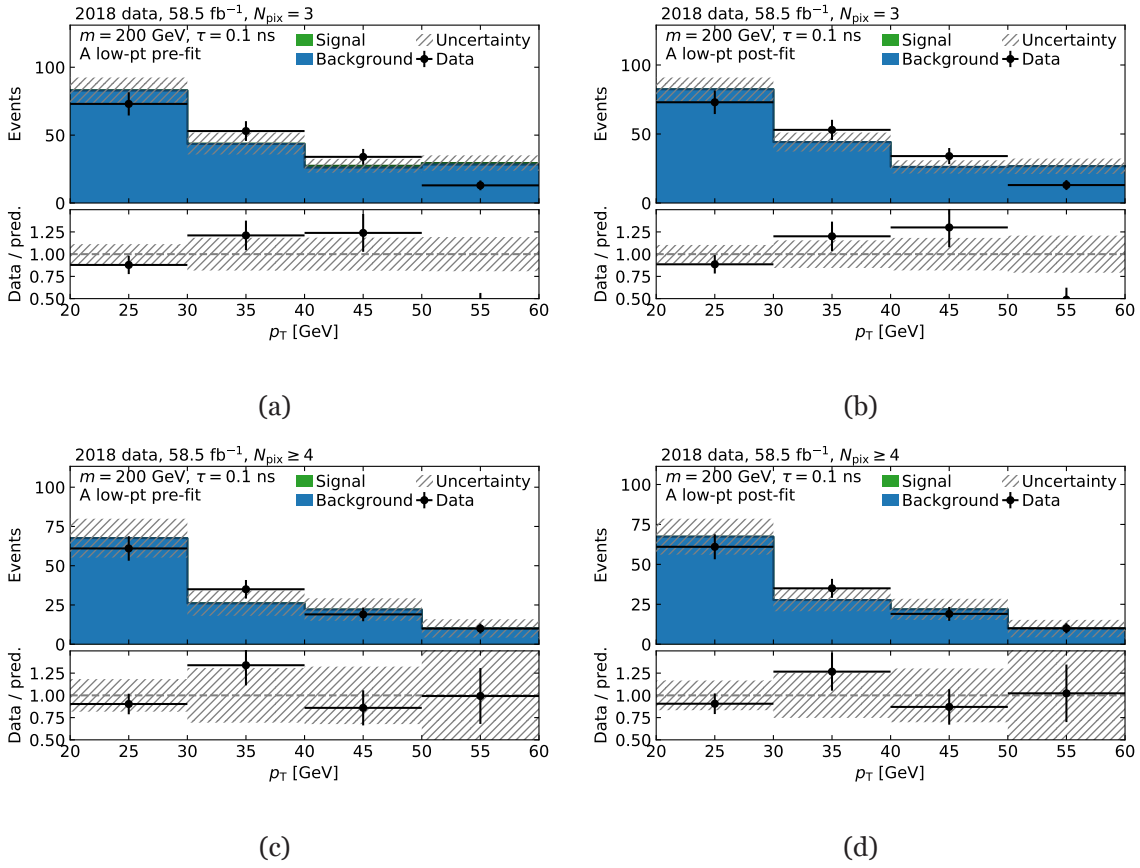


Figure B.23.: Transverse momentum distributions pre- and post-fit in the A_L region. The fit is performed for three- and four-layer distributions separately. Shown are (pseudo-)data, background estimate and a signal hypothesis for $\tau(\tilde{\chi}_1^{\pm}) = 0.1\,\text{ns}$ and $m = 200\,\text{GeV}$. Statistical and systematic uncertainties are displayed. The lower panel shows the ratio of (pseudo-)data and signal plus background prediction, as well as relative uncertainties.

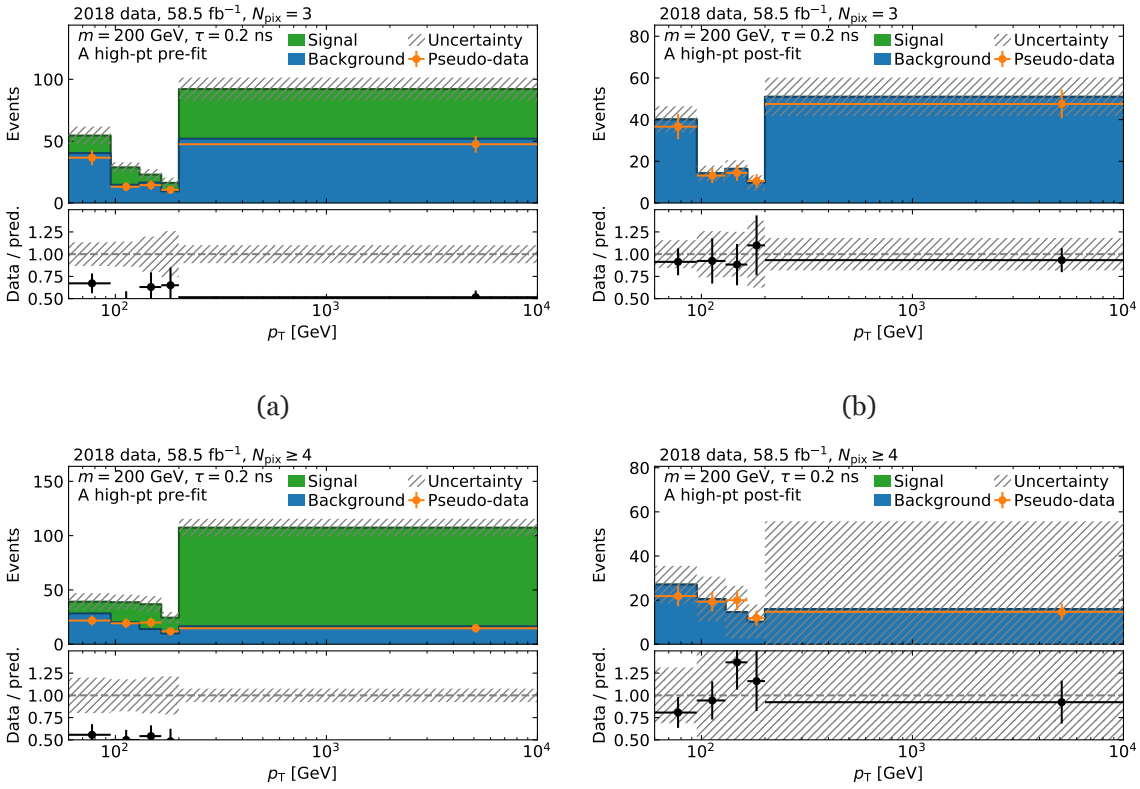


Figure B.24.: Transverse momentum distributions pre- and post-fit in the A_H region. The fit is performed for three- and four-layer distributions separately. Shown are (pseudo-)data, background estimate and a signal hypothesis for $\tau(\tilde{\chi}_1^{\pm}) = 0.2\,\text{ns}$ and $m = 200\,\text{GeV}$. Statistical and systematic uncertainties are displayed. The lower panel shows the ratio of (pseudo-)data and signal plus background prediction, as well as relative uncertainties.

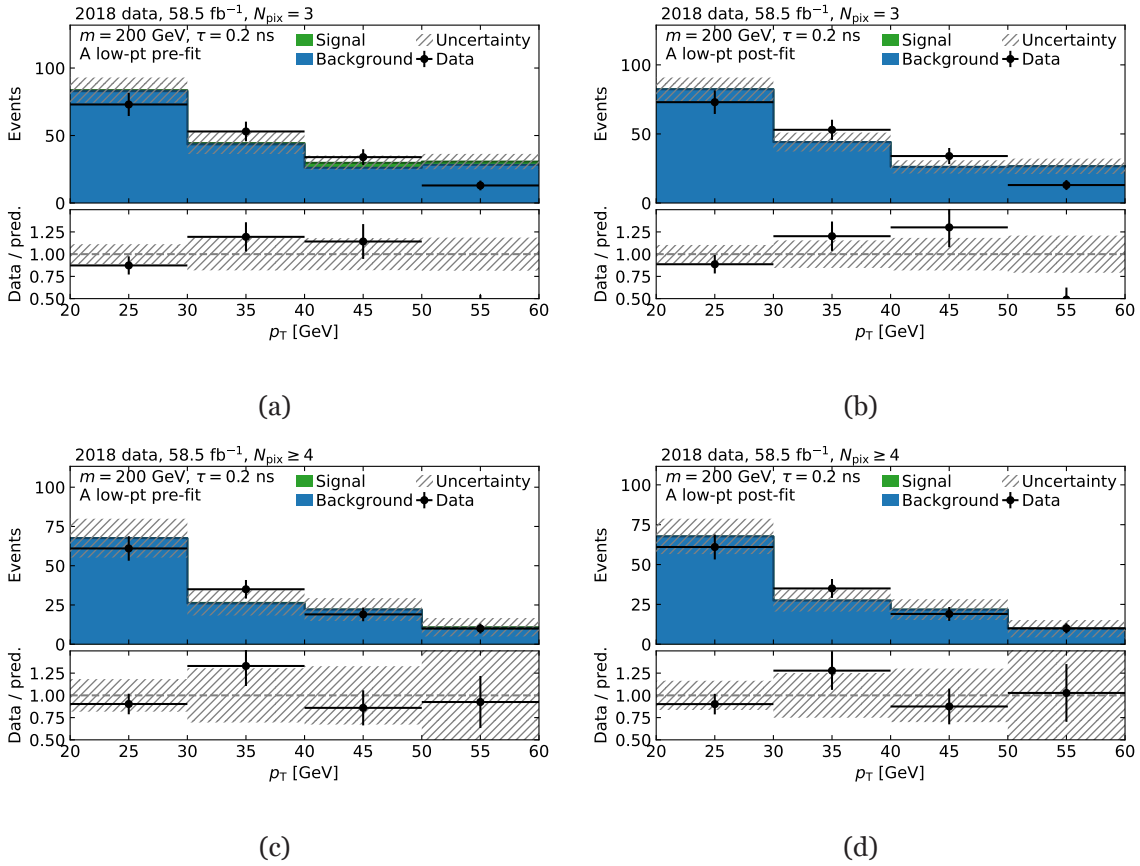


Figure B.25.: Transverse momentum distributions pre- and post-fit in the A_L region. The fit is performed for three- and four-layer distributions separately. Shown are (pseudo-)data, background estimate and a signal hypothesis for $\tau(\tilde{\chi}_1^{\pm}) = 0.2\,\text{ns}$ and $m = 200\,\text{GeV}$. Statistical and systematic uncertainties are displayed. The lower panel shows the ratio of (pseudo-)data and signal plus background prediction, as well as relative uncertainties.

B.2.2. Combined fit

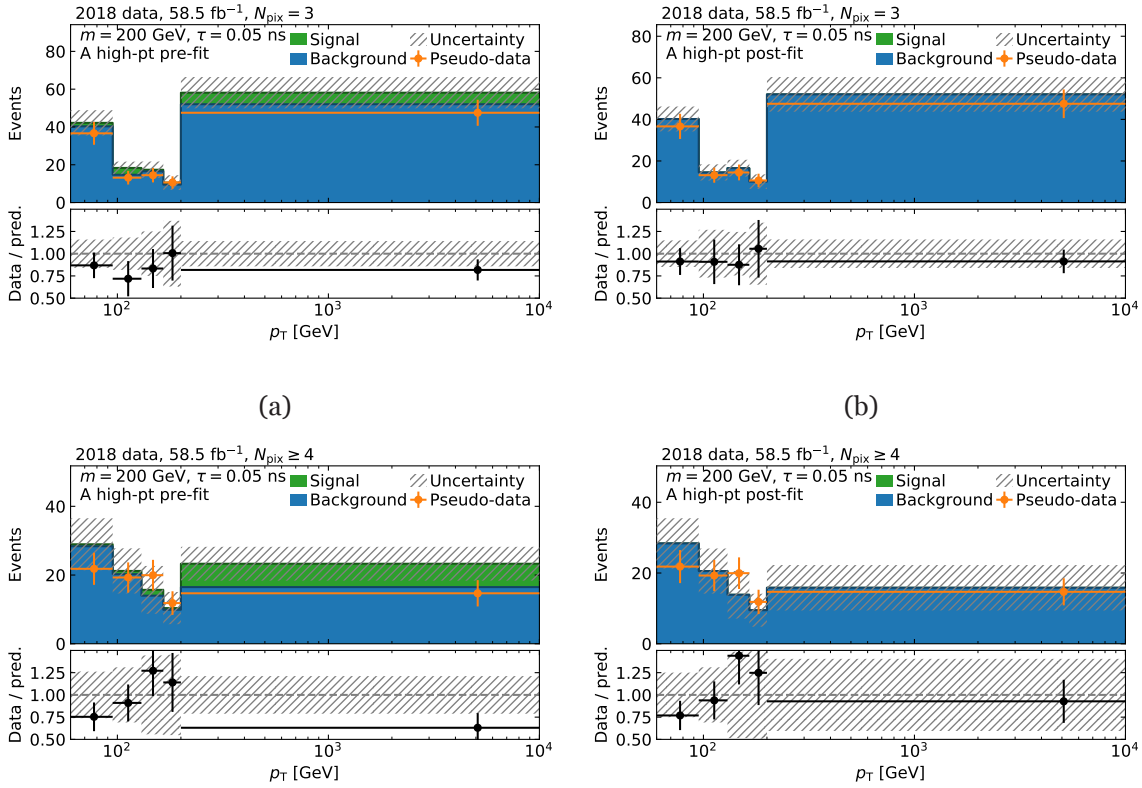


Figure B.26.: Transverse momentum distributions pre- and post-fit in the A_H region. The fit is performed simultaneously for three- and four-layer distributions. Shown are (pseudo-)data, background estimate and a signal hypothesis for $\tau(\tilde{\chi}_1^{\pm}) = 0.05$ ns and $m = 200$ GeV. Statistical and systematic uncertainties are displayed. The lower panel shows the ratio of (pseudo-)data and signal plus background prediction, as well as relative uncertainties.

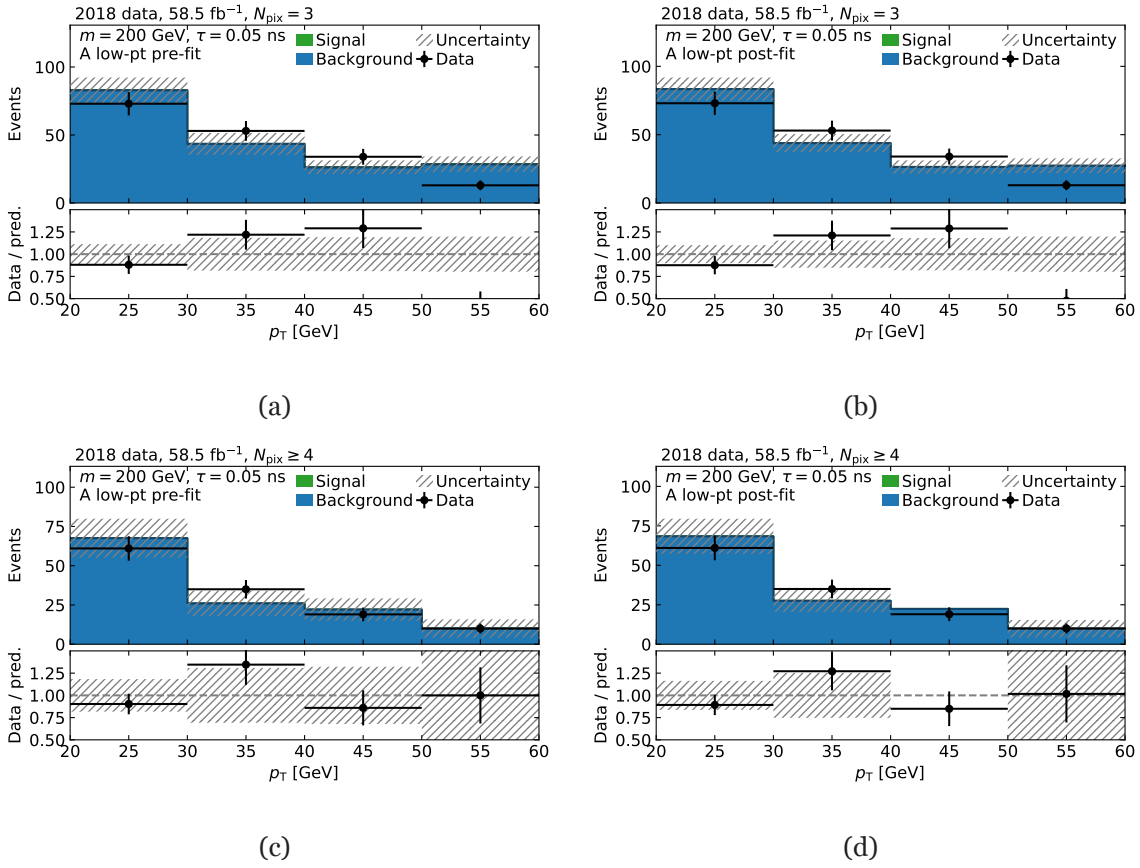


Figure B.27.: Transverse momentum distributions pre- and post-fit in the A_L region. The fit is performed simultaneously for three- and four-layer distributions. Shown are (pseudo-)data, background estimate and a signal hypothesis for $\tau(\tilde{\chi}_1^{\pm}) = 0.05$ ns and $m = 200$ GeV. Statistical and systematic uncertainties are displayed. The lower panel shows the ratio of (pseudo-)data and signal plus background prediction, as well as relative uncertainties.

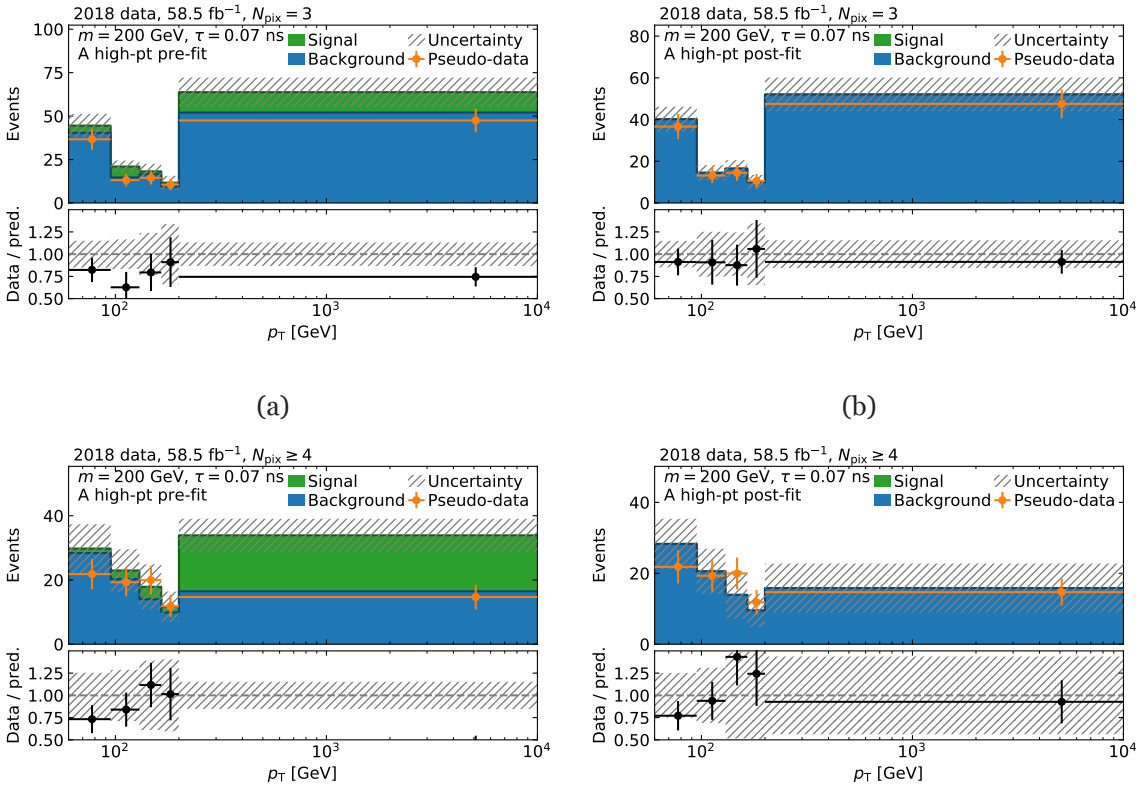


Figure B.28.: Transverse momentum distributions pre- and post-fit in the A_H region. The fit is performed simultaneously for three- and four-layer distributions. Shown are (pseudo-)data, background estimate and a signal hypothesis for $\tau(\tilde{\chi}_1^{\pm}) = 0.07\,\text{ns}$ and $m = 200\,\text{GeV}$. Statistical and systematic uncertainties are displayed. The lower panel shows the ratio of (pseudo-)data and signal plus background prediction, as well as relative uncertainties.

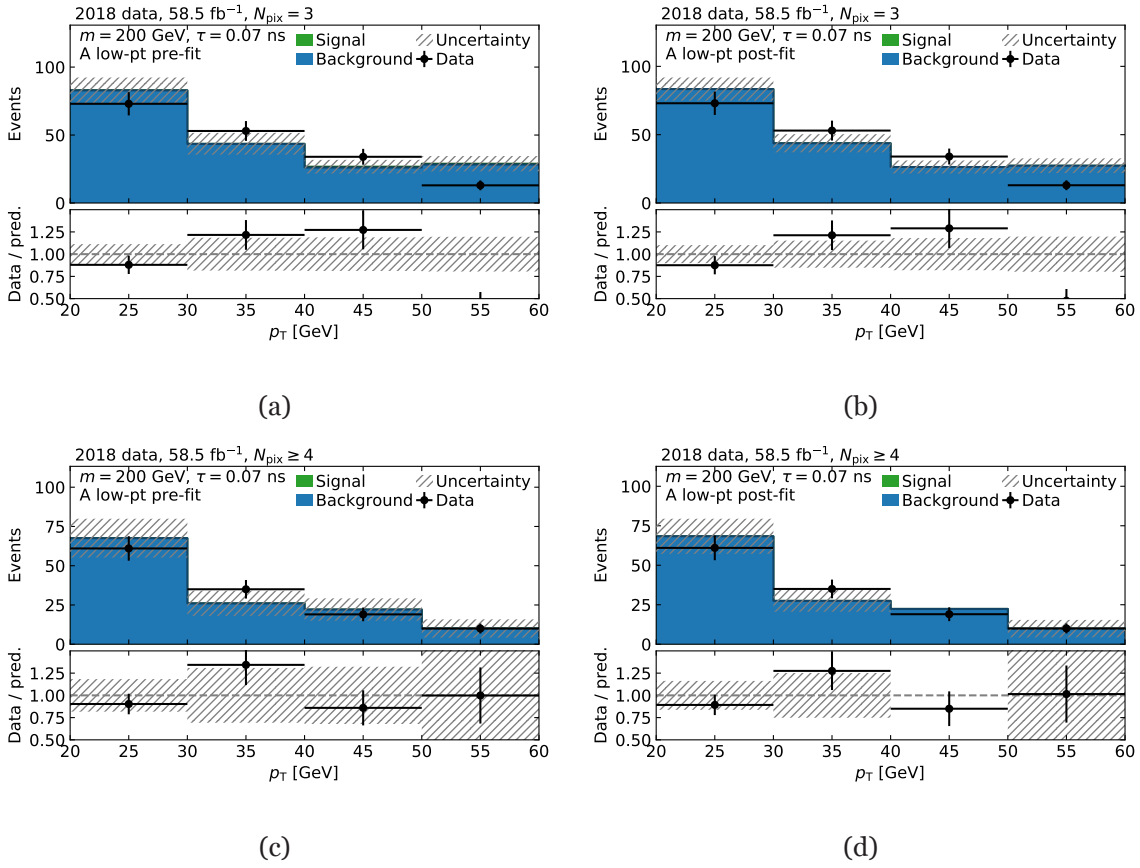


Figure B.29.: Transverse momentum distributions pre- and post-fit in the A_L region. The fit is performed simultaneously for three- and four-layer distributions. Shown are (pseudo-)data, background estimate and a signal hypothesis for $\tau(\tilde{\chi}_1^{\pm}) = 0.07$ ns and $m = 200$ GeV. Statistical and systematic uncertainties are displayed. The lower panel shows the ratio of (pseudo-)data and signal plus background prediction, as well as relative uncertainties.

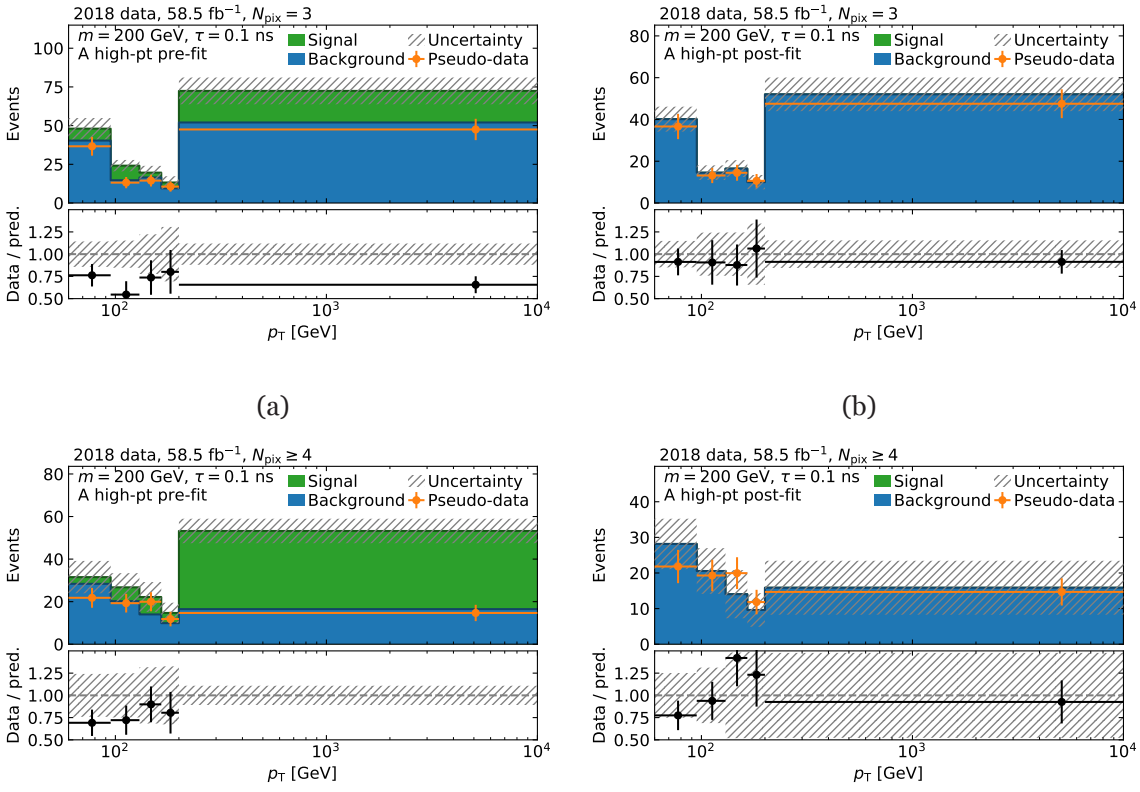


Figure B.30.: Transverse momentum distributions pre- and post-fit in the A_H region. The fit is performed simultaneously for three- and four-layer distributions. Shown are (pseudo-)data, background estimate and a signal hypothesis for $\tau(\tilde{\chi}_1^{\pm}) = 0.1\,\text{ns}$ and $m = 200\,\text{GeV}$. Statistical and systematic uncertainties are displayed. The lower panel shows the ratio of (pseudo-)data and signal plus background prediction, as well as relative uncertainties.

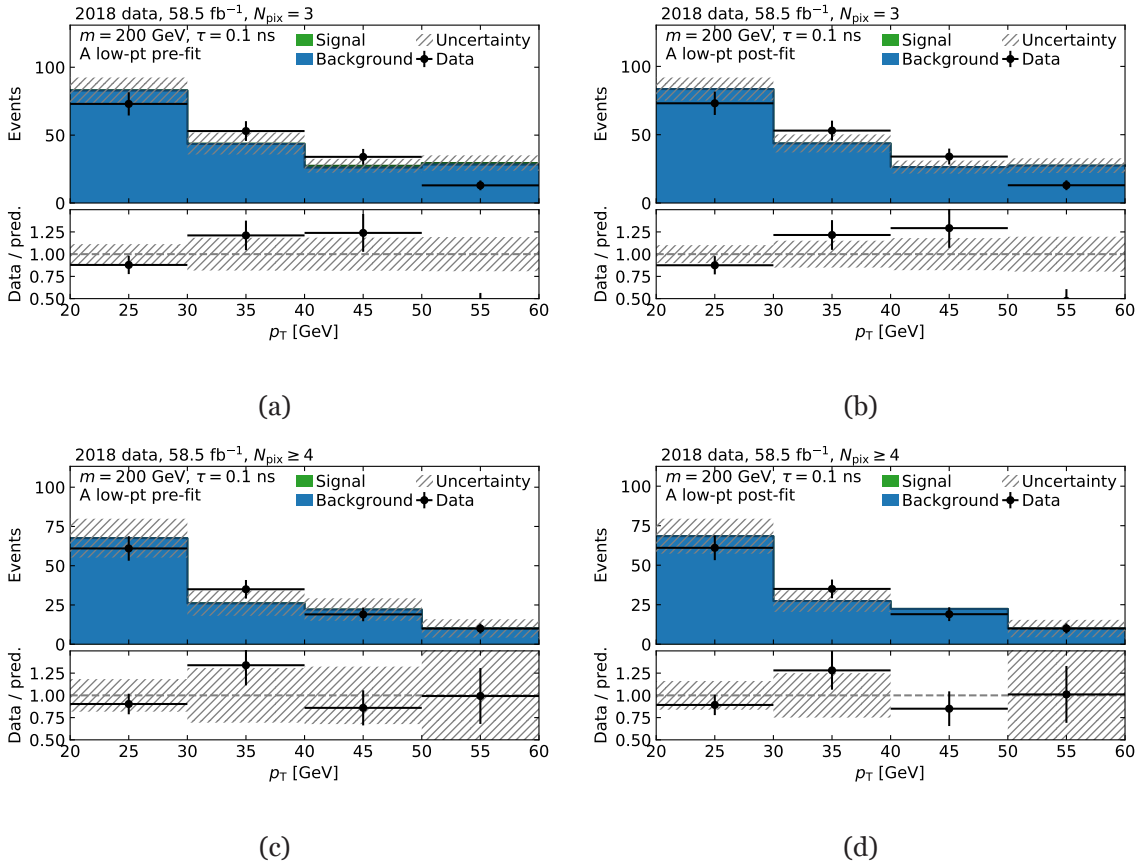


Figure B.31.: Transverse momentum distributions pre- and post-fit in the A_L region. The fit is performed simultaneously for three- and four-layer distributions. Shown are (pseudo-)data, background estimate and a signal hypothesis for $\tau(\tilde{\chi}_1^{\pm}) = 0.1\,\text{ns}$ and $m = 200\,\text{GeV}$. Statistical and systematic uncertainties are displayed. The lower panel shows the ratio of (pseudo-)data and signal plus background prediction, as well as relative uncertainties.

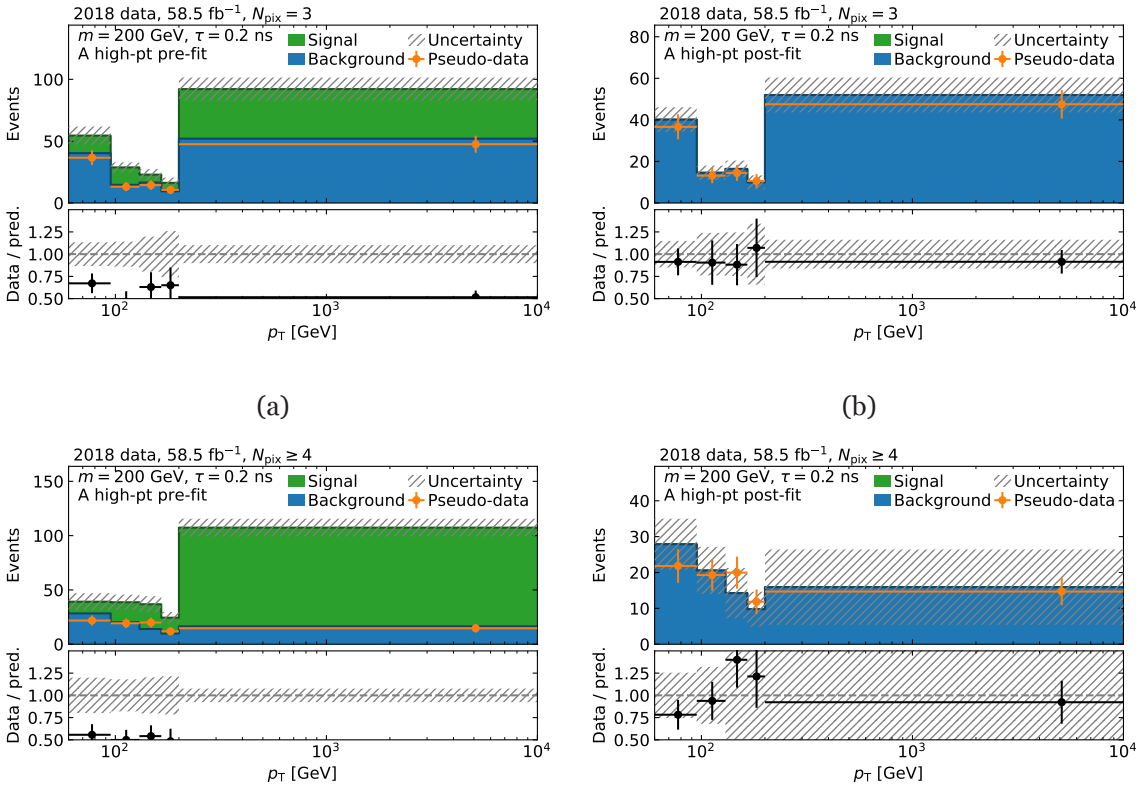


Figure B.32.: Transverse momentum distributions pre- and post-fit in the A_H region. The fit is performed simultaneously for three- and four-layer distributions. Shown are (pseudo-)data, background estimate and a signal hypothesis for $\tau(\tilde{\chi}_1^{\pm}) = 0.2$ ns and $m = 200$ GeV. Statistical and systematic uncertainties are displayed. The lower panel shows the ratio of (pseudo-)data and signal plus background prediction, as well as relative uncertainties.

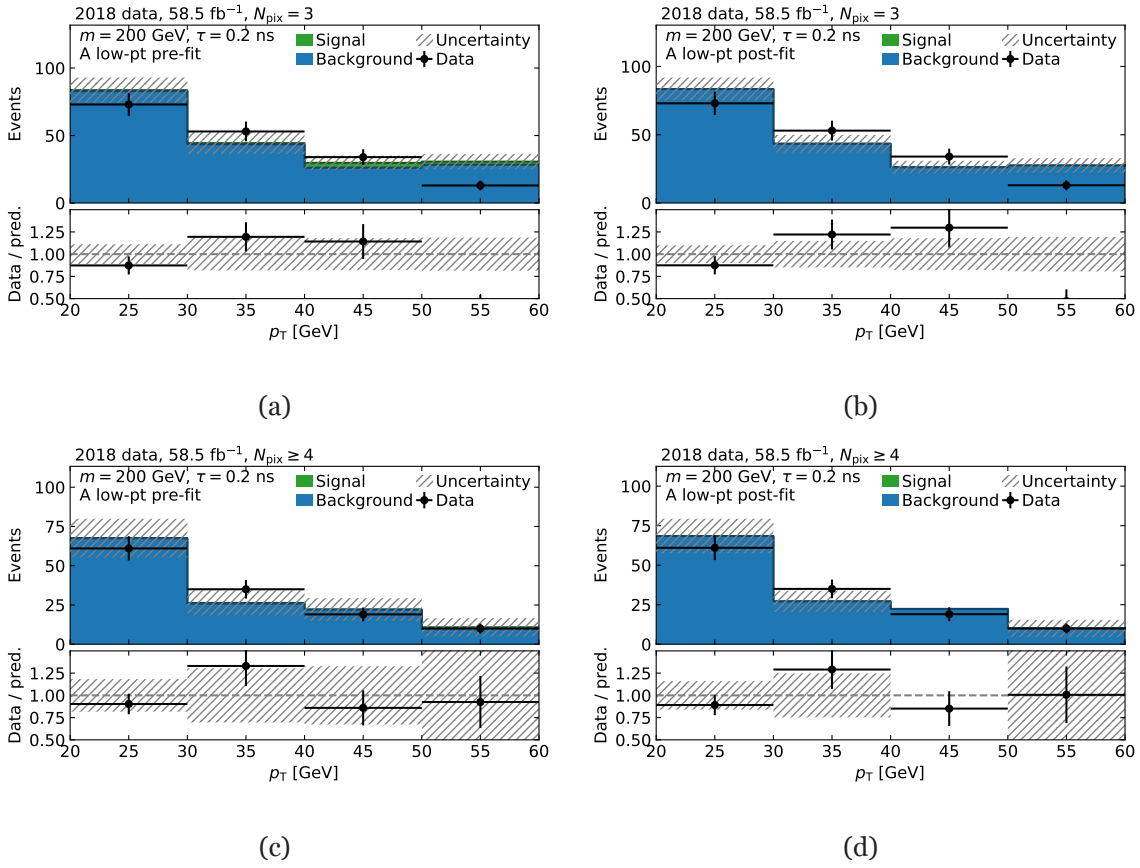
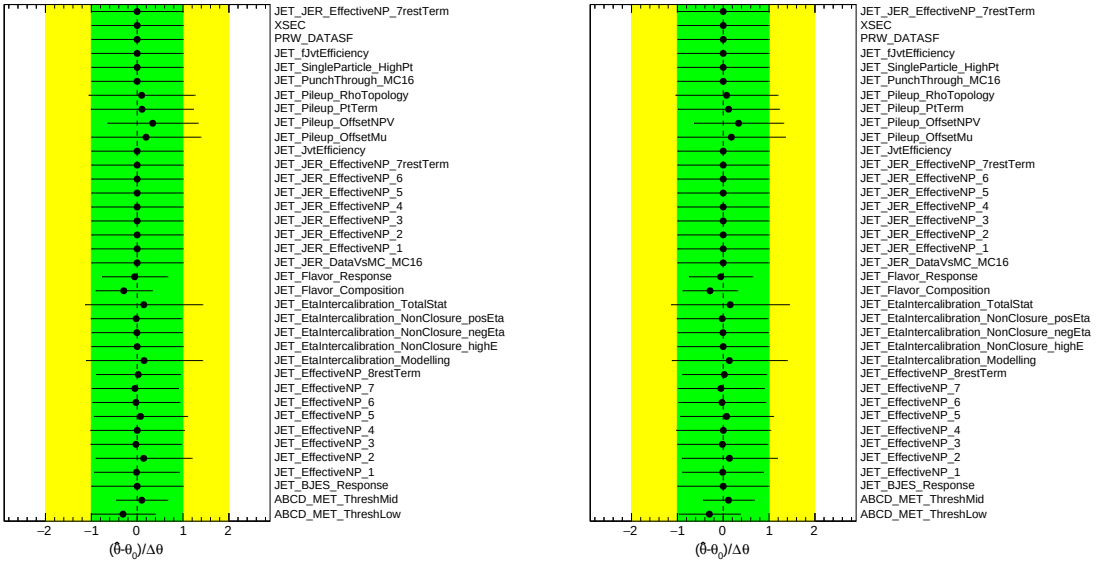


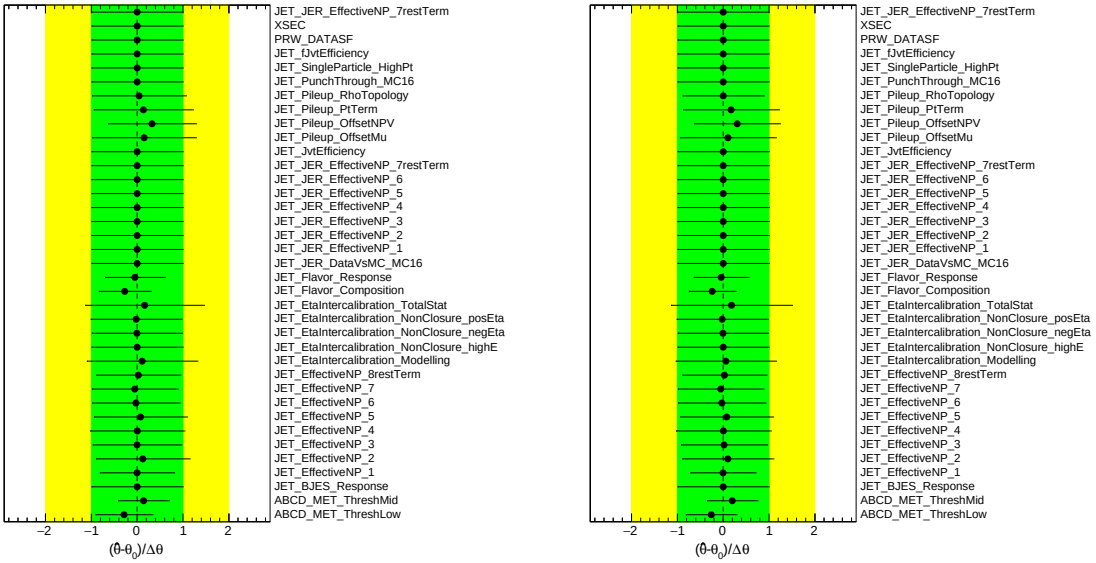
Figure B.33.: Transverse momentum distributions pre- and post-fit in the A_L region. The fit is performed simultaneously for three- and four-layer distributions. Shown are (pseudo-)data, background estimate and a signal hypothesis for $\tau(\tilde{\chi}_1^{\pm}) =$ 0.2 ns and $m = 200$ GeV. Statistical and systematic uncertainties are displayed. The lower panel shows the ratio of (pseudo-)data and signal plus background prediction, as well as relative uncertainties.

B.3. Nuisance parameter pulls



(a) $\tau = 0.05\,\text{ns}$

(b) $\tau = 0.07\,\text{ns}$

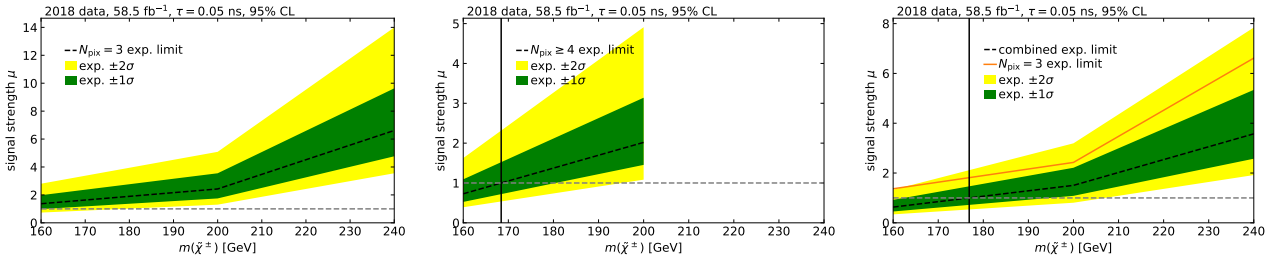


(c) $\tau = 0.1\,\text{ns}$

(d) $\tau = 0.2\,\text{ns}$

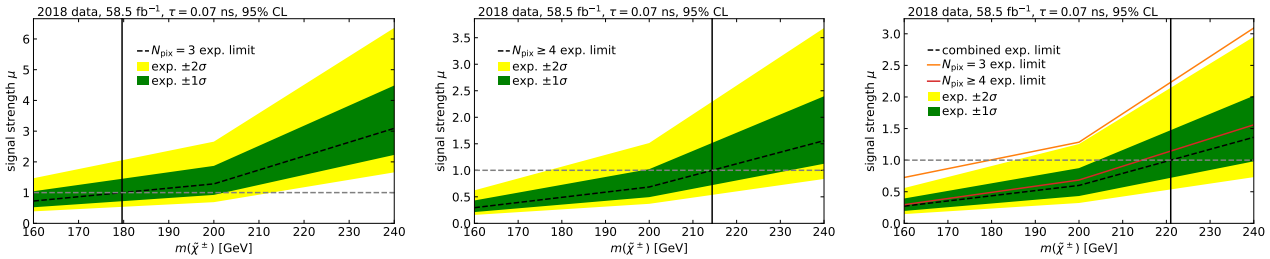
Figure B.34.: Example pull plots for nuisance parameters in a maximum likelihood fit to a signal hypothesis of $m(\tilde{\chi}_1^{\pm}) = 200\,\text{GeV}$ and four lifetimes.

B.4. Signal strength upper limits



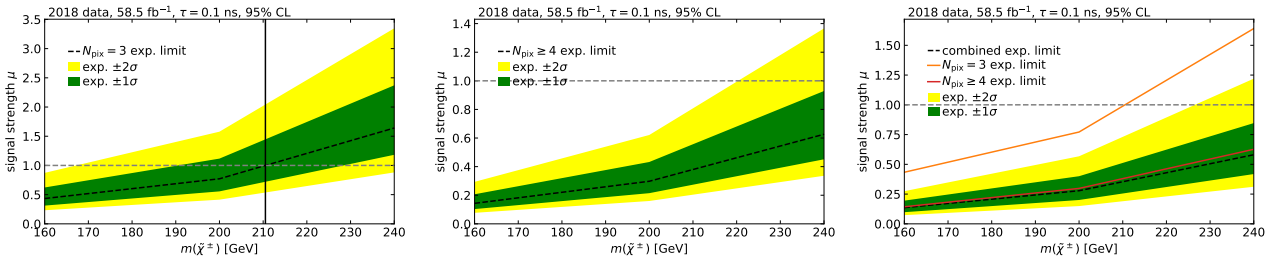
(a) (b) (c)

Figure B.35.: Upper limit signal strength plots for $\tau(\tilde{\chi}_1^{\pm}) = 0.05$ ns. Three-layer, four-layer and a combined upper-limit are shown. Missing limits at a mass point indicate the limit calculation terminating abnormally.



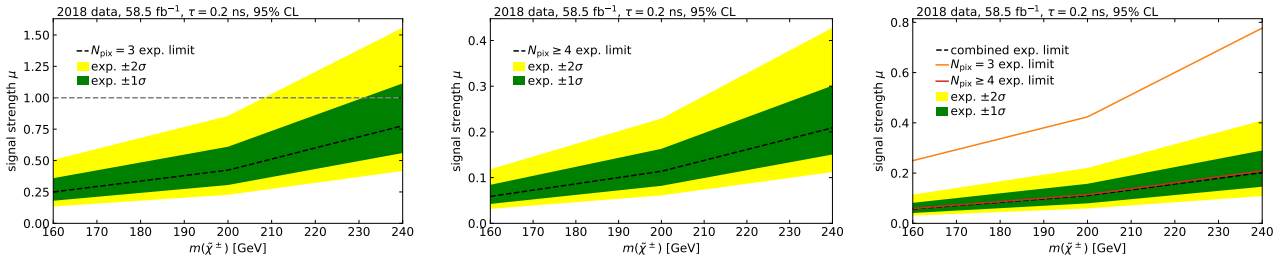
(a) (b) (c)

Figure B.36.: Upper limit signal strength plots for $\tau(\tilde{\chi}_1^{\pm}) = 0.07$ ns. Three-layer, four-layer and a combined upper-limit are shown. Missing limits at a mass point indicate the limit calculation terminating abnormally.



(a) (b) (c)

Figure B.37.: Upper limit signal strength plots for $\tau(\tilde{\chi}_1^{\pm}) = 0.1$ ns. Three-layer, four-layer and a combined upper-limit are shown. Missing limits at a mass point indicate the limit calculation terminating abnormally.



(a) (b) (c)

Figure B.38.: Upper limit signal strength plots for $\tau(\tilde{\chi}_1^{\pm}) = 0.2\,\text{ns}$. Three-layer, four-layer and a combined upper-limit are shown. Missing limits at a mass point indicate the limit calculation terminating abnormally.

Bibliography

[1] S. Weinberg. “A Model of Leptons”. In: *Phys. Rev. Lett.* 19 (1967), pp. 1264–1266. DOI: 10.1103/PhysRevLett.19.1264.

[2] S. L. Glashow. “Partial Symmetries of Weak Interactions”. In: *Nucl. Phys.* 22 (1961), pp. 579–588. DOI: 10.1016/0029-5582(61)90469-2.

[3] A. Salam. “Weak and Electromagnetic Interactions”. In: *Conf. Proc.* C680519 (1968), pp. 367–377. DOI: 10.1142/9789812795915_0034.

[4] G. ’t Hooft and M. Veltman. “Regularization and Renormalization of Gauge Fields”. In: *Nucl. Phys. B* 44 (1972), pp. 189–213. DOI: 10.1016/0550-3213(72)90279-9.

[5] P. Gessinger-Befurt. “Search for new heavy charged gauge bosons with the ATLAS detector”. Presented 15 May 2017. Apr. 2017. URL: https://cds.cern.ch/record/2276657.

[6] D. Griffiths. *Introduction to elementary particles*. Wiley-VCH, 2008. ISBN: 978-3-527-40601-2.

[7] J. J. Thomson. “XL. Cathode rays”. In: *Phil. Mag. Journ. Sci.* 44.269 (1897), pp. 293–316. DOI: 10.1080/14786449708621070.

[8] CMS Collaboration. “Observation of a new boson at a mass of 125 GeV with the CMS experiment at the LHC”. In: *Phys. Lett.* B716 (2012), pp. 30–61. DOI: 10.1016/j.physletb.2012.08.021. arXiv: 1207.7235 [hep-ex].

[9] ATLAS Collaboration. “Observation of a new particle in the search for the Standard Model Higgs boson with the ATLAS detector at the LHC”. In: *Phys. Lett.* B716 (2012), pp. 1–29. DOI: 10.1016/j.physletb.2012.08.020. arXiv: 1207.7214 [hep-ex].

[10] P. W. Higgs. “Broken Symmetries and the Masses of Gauge Bosons”. In: *Phys. Rev. Lett.* 13 (1964). Ed. by J. Taylor, pp. 508–509. DOI: 10.1103/PhysRevLett.13.508.

[11] P. W. Higgs. “Broken symmetries, massless particles and gauge fields”. In: *Phys. Lett.* 12 (1964), pp. 132–133. DOI: 10.1016/0031-9163(64)91136-9.

[12] F. Englert and R. Brout. “Broken Symmetry and the Mass of Gauge Vector Mesons”. In: *Phys. Rev. Lett.* 13 (1964). Ed. by J. Taylor, pp. 321–323. DOI: 10.1103/PhysRevLett.13.321.

[13] G. Guralnik, C. Hagen, and T. Kibble. “Global Conservation Laws and Massless Particles”. In: *Phys. Rev. Lett.* 13 (1964). Ed. by J. Taylor, pp. 585–587. DOI: 10.1103/PhysRevLett.13.585.

[14] P. W. Higgs. “Spontaneous Symmetry Breakdown without Massless Bosons”. In: *Phys. Rev.* 145 (1966), pp. 1156–1163. DOI: 10.1103/PhysRev.145.1156.

[15] T. Kibble. “Symmetry breaking in nonAbelian gauge theories”. In: *Phys. Rev.* 155 (1967). Ed. by J. Taylor, pp. 1554–1561. DOI: 10.1103/PhysRev.155.1554.

[16] A. Einstein. “Über einen die Erzeugung und Verwandlung des Lichtes betreffenden heuristischen Gesichtspunkt”. In: *Annalen der Physik* 322.6 (1905), pp. 132–148. DOI: https://doi.org/10.1002/andp.19053220607. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/andp.19053220607. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/andp.19053220607.

[17] H. Yukawa. “On the Interaction of Elementary Particles I”. In: *Proc. Phys. Math. Soc. Jap.* 17 (1935), pp. 48–57. DOI: 10.1143/PTPS.1.1.

[18] P. A. Dirac. “The quantum theory of the electron”. In: *Proc. Roy. Soc. Lond. A* 117 (1928), pp. 610–624. DOI: 10.1098/rspa.1928.0023.

[19] C. D. Anderson. “Energies of Cosmic-Ray Particles”. In: *Phys. Rev.* 41 (1932), pp. 405–421. DOI: 10.1103/PhysRev.41.405.

[20] W. Pauli. “Dear radioactive ladies and gentlemen”. In: *Phys. Today* 31N9 (1978), p. 27.

[21] C. Cowan et al. “Detection of the free neutrino: A Confirmation”. In: *Science* 124 (1956), pp. 103–104. DOI: 10.1126/science.124.3212.103.

[22] E. Fermi. “An attempt of a theory of beta radiation. 1.” In: *Z. Phys.* 88 (1934), pp. 161–177. DOI: 10.1007/BF01351864.

[23] T. Nakano and K. Nishijima. “Charge Independence for V-particles”. In: *Prog. Theor. Phys.* 10 (1953), pp. 581–582. DOI: 10.1143/PTP.10.581.

[24] A. Seriff et al. “Cloud-Chamber Observations of the New Unstable Cosmic-Ray Particles”. In: *Phys. Rev.* 78.3 (1950), p. 290. DOI: 10.1103/PhysRev.78.290.

[25] G. Rochester and C. Butler. “Evidence for the Existence of New Unstable Elementary Particles”. In: *Nature* 160 (1947), pp. 855–857. DOI: 10.1038/160855a0.

[26] M. Gell-Mann. “Isotopic Spin and New Unstable Particles”. In: *Phys. Rev.* 92 (1953), pp. 833–834. DOI: 10.1103/PhysRev.92.833.

[27] M. Gell-Mann and Y. Ne’eman. “The Eightfold way: a review with a collection of reprints”. In: (Sept. 1964).

[28] M. Gell-Mann. “A Schematic Model of Baryons and Mesons”. In: *Phys. Lett.* 8 (1964), pp. 214–215. DOI: 10.1016/S0031-9163(64)92001-3.

[29] G. Zweig. “An SU(3) model for strong interaction symmetry and its breaking. Version 2”. In: *Developments in the quark theory of hadrons, Vol. 1. 1964 - 1978*. Ed. by D. Lichtenberg and S. P. Rosen. Feb. 1964, pp. 22–101.

[30] R. Feynman. “Mathematical formulation of the quantum theory of electromagnetic interaction”. In: *Phys. Rev.* 80 (1950). Ed. by L. Brown, pp. 440–457. DOI: 10.1103/PhysRev.80.440.

[31] S. Tomonaga. “On a relativistically invariant formulation of the quantum theory of wave fields”. In: *Prog. Theor. Phys.* 1 (1946), pp. 27–42. DOI: 10.1143/PTP.1.27.

[32] J. S. Schwinger. “On Quantum electrodynamics and the magnetic moment of the electron”. In: *Phys. Rev.* 73 (1948), pp. 416–417. DOI: 10.1103/PhysRev.73.416.

[33] J. S. Schwinger. “Quantum electrodynamics. I A covariant formulation”. In: *Phys. Rev.* 74 (1948). Ed. by K. Milton, p. 1439. DOI: 10.1103/PhysRev.74.1439.

[34] R. Feynman. “Space - time approach to quantum electrodynamics”. In: *Phys. Rev.* 76 (1949). Ed. by L. Brown, pp. 769–789. DOI: 10.1103/PhysRev.76.769.

[35] R. Feynman. “The Theory of positrons”. In: *Phys. Rev.* 76 (1949). Ed. by L. Brown, pp. 749–759. DOI: 10.1103/PhysRev.76.749.

[36] C.-N. Yang and R. L. Mills. “Conservation of Isotopic Spin and Isotopic Gauge Invariance”. In: *Phys. Rev.* 96 (1954). Ed. by J.-P. Hsu and D. Fine, pp. 191–195. DOI: 10.1103/PhysRev.96.191.

[37] M. L. Perl et al. “Evidence for Anomalous Lepton Production in e+ - e- Annihilation”. In: *Phys. Rev. Lett.* 35 (1975), pp. 1489–1492. DOI: 10.1103/PhysRevLett.35.1489.

[38] R. Schwitters. “Fundamental Particles with Charm”. In: *Sci. Am.* 237N4 (1977), pp. 56–70. DOI: 10.1038/scientificamerican1077-56.

[39] M. Basile et al. “Evidence for a New Particle With Naked ’Beauty’ and for Its Associated Production in High-energy (*pp*) Interactions”. In: *Lett. Nuovo Cim.* 31 (1981), p. 97. DOI: 10.1007/BF02822406.

[40] D. Drijard et al. “Further Investigation of Beauty Baryon Production at the {ISR}”. In: *Phys. Lett. B* 108 (1982), pp. 361–366. DOI: 10.1016/0370-2693(82)91213-8.

[41] CDF Collaboration. “Observation of top quark production in $\bar{p}p$ collisions”. In: *Phys. Rev. Lett.* 74 (1995), pp. 2626–2631. DOI: 10.1103/PhysRevLett.74.2626. arXiv: hep-ex/9503002.

[42] DØ Collaboration. “Observation of the top quark”. In: *Phys. Rev. Lett.* 74 (1995), pp. 2632–2637. DOI: 10.1103/PhysRevLett.74.2632. arXiv: hep-ex/9503003.

[43] UA1 Collaboration. “Experimental Observation of Isolated Large Transverse Energy Electrons with Associated Missing Energy at s**(1/2) = 540-GeV”. In: *Phys. Lett. B* 122 (1983), pp. 103–116. DOI: 10.1016/0370-2693(83)91177-2.

[44] UA1 Collaboration. “Experimental Observation of Lepton Pairs of Invariant Mass Around 95-GeV/c**2 at the CERN SPS Collider”. In: *Phys. Lett. B* 126 (1983), pp. 398–410. DOI: 10.1016/0370-2693(83)90188-0.

[45] KATRIN Collaboration. “Improved Upper Limit on the Neutrino Mass from a Direct Kinematic Method by KATRIN”. In: *Phys. Rev. Lett.* 123.22 (2019), p. 221802. DOI: 10.1103/PhysRevLett.123.221802. arXiv: 1909.06048 [hep-ex].

[46] Particle Data Group Collaboration. “Review of Particle Physics”. In: *PTEP* 2020.8 (Aug. 2020). 083C01. DOI: 10.1093/ptep/ptaa104.

[47] L. H. Ryder. *Quantum Field Theory*. Cambridge University Press, 1996. ISBN: 978-0-521-47814-4.

[48] F. Siegert. “Monte-Carlo event generation for the LHC”. PhD thesis. Durham U., 2010. URL: http://cds.cern.ch/record/1600005.

[49] J. M. Campbell, J. Huston, and W. Stirling. “Hard Interactions of Quarks and Gluons: A Primer for LHC Physics”. In: *Rept. Prog. Phys.* 70 (2007), p. 89. DOI: 10.1088/0034-4885/70/1/R02. arXiv: hep-ph/0611148.

[50] J. C. Collins, D. E. Soper, and G. F. Sterman. “Factorization of Hard Processes in QCD”. In: *Adv. Ser. Direct. High Energy Phys.* 5 (1989), pp. 1–91. DOI: 10.1142/9789814503266_0001. arXiv: hep-ph/0409313 [hep-ph].

[51] G. Altarelli and G. Parisi. “Asymptotic Freedom in Parton Language”. In: *Nucl. Phys.* B126 (1977), pp. 298–318. DOI: 10.1016/0550-3213(77)90384-4.

[52] Y. L. Dokshitzer. “Calculation of the Structure Functions for Deep Inelastic Scattering and e+ e- Annihilation by Perturbation Theory in Quantum Chromodynamics.” In: *Sov. Phys. JETP* 46 (1977). [Zh. Eksp. Teor. Fiz.73,1216(1977)], pp. 641–653.

[53] V. N. Gribov and L. N. Lipatov. “Deep inelastic e p scattering in perturbation theory”. In: *Sov. J. Nucl. Phys.* 15 (1972). [Yad. Fiz.15,781(1972)], pp. 438–450.

[54] A. Martin et al. “Parton distributions for the LHC”. In: *Eur. Phys. J. C* 63 (2009), pp. 189–285. DOI: 10.1140/epjc/s10052-009-1072-5. arXiv: 0901.0002 [hep-ph].

[55] T. Lee and C.-N. Yang. “Question of Parity Conservation in Weak Interactions”. In: *Phys. Rev.* 104 (1956), pp. 254–258. DOI: 10.1103/PhysRev.104.254.

[56] C. Wu et al. “Experimental Test of Parity Conservation in β Decay”. In: *Phys. Rev.* 105 (1957), pp. 1413–1414. DOI: 10.1103/PhysRev.105.1413.

[57] F. Zwicky. “On the Masses of Nebulae and of Clusters of Nebulae”. In: *Astrophys. J.* 86 (1937), pp. 217–246. DOI: 10.1086/143864.

[58] V. C. Rubin and J. Ford W.Kent. “Rotation of the Andromeda Nebula from a Spectroscopic Survey of Emission Regions”. In: *Astrophys. J.* 159 (1970), pp. 379–403. DOI: 10.1086/150317.

[59] M. S. Roberts and R. N. Whitehurst. “The rotation curve and geometry of M31 at large galactocentric distances.” In: *Astrophys. J.* 201 (1975), pp. 327–346. DOI: 10.1086/153889.

[60] A. A. Penzias and R. W. Wilson. “A Measurement of excess antenna temperature at 4080-Mc/s”. In: *Astrophys. J.* 142 (1965), pp. 419–421. DOI: 10.1086/148307.

[61] R. Dicke et al. “Cosmic Black-Body Radiation”. In: *Astrophys. J.* 142 (1965), pp. 414–419. DOI: 10.1086/148306.

[62] L. Susskind. “Dynamics of Spontaneous Symmetry Breaking in the Weinberg-Salam Theory”. In: *Phys. Rev. D* 20 (1979), pp. 2619–2625. DOI: 10.1103/PhysRevD.20.2619.

[63] E. Gildener. “Gauge Symmetry Hierarchies”. In: *Phys. Rev. D* 14 (1976), p. 1667. DOI: 10.1103/PhysRevD.14.1667.

[64] S. Weinberg. “Implications of Dynamical Symmetry Breaking”. In: *Phys. Rev. D* 13 (1976). [Addendum: Phys.Rev.D 19, 1277–1280 (1979)], pp. 974–996. DOI: 10.1103/PhysRevD.19.1277.

[65] S. Weinberg. “Implications of dynamical symmetry breaking: An addendum”. In: *Phys. Rev. D* 19 (4 Feb. 1979), pp. 1277–1280. DOI: 10.1103/PhysRevD.19.1277. URL: https://link.aps.org/doi/10.1103/PhysRevD.19.1277.

[66] G. ’t Hooft et al., eds. *Recent Developments in Gauge Theories. Proceedings, Nato Advanced Study Institute, Cargese, France, August 26 - September 8, 1979*. Vol. 59. 1980, pp.1–438. DOI: 10.1007/978-1-4684-7571-5.

[67] P. Ramond. “Dual Theory for Free Fermions”. In: *Phys. Rev. D* 3 (1971), pp. 2415–2418. DOI: 10.1103/PhysRevD.3.2415.

[68] A. Neveu and J. Schwarz. “Factorizable dual model of pions”. In: *Nucl. Phys. B* 31 (1971), pp. 86–112. DOI: 10.1016/0550-3213(71)90448-2.

[69] J.-L. Gervais and B. Sakita. “Field Theory Interpretation of Supergauges in Dual Models”. In: *Nucl. Phys. B* 34 (1971). Ed. by K. Kikkawa, M. Virasoro, and S. Wadia, pp. 632–639. DOI: 10.1016/0550-3213(71)90351-8.

[70] Y. A. Golfand and E. P. Likhtman. “Extension of the Algebra of Poincare Group Generators and Violation of p Invariance”. In: *JETP Lett.* 13 (1971), pp. 323–326.

[71] J. Wess and B. Zumino. “Supergauge Transformations in Four-Dimensions”. In: *Nucl. Phys. B* 70 (1974). Ed. by A. Salam and E. Sezgin, pp. 39–50. DOI: 10.1016/0550-3213(74)90355-1.

[72] D. V. Volkov and V. P. Akulov. “Is the Neutrino a Goldstone Particle?” In: *Phys. Lett. B* 46 (1973), pp. 109–110. DOI: 10.1016/0370-2693(73)90490-5.

[73] S. P. Martin. *A Supersymmetry primer*. 1997. DOI: 10.1142/9789812839657_0001. arXiv: hep-ph/9709356 [hep-ph].

[74] J. Wess and B. Zumino. “Supergauge Invariant Extension of Quantum Electrodynamics”. In: *Nucl. Phys. B* 78 (1974), p. 1. DOI: 10.1016/0550-3213(74)90112-6.

[75] A. Salam and J. A. Strathdee. “Supersymmetry and Nonabelian Gauges”. In: *Phys. Lett. B* 51 (1974), pp. 353–355. DOI: 10.1016/0370-2693(74)90226-3.

[76] S. Ferrara and B. Zumino. “Supergauge Invariant Yang-Mills Theories”. In: *Nucl. Phys. B* 79 (1974), p. 413. DOI: 10.1016/0550-3213(74)90559-8.

[77] G. R. Farrar and P. Fayet. “Phenomenology of the Production, Decay, and Detection of New Hadronic States Associated with Supersymmetry”. In: *Phys. Lett. B* 76 (1978), pp. 575–579. DOI: 10.1016/0370-2693(78)90858-4.

[78] L. Randall and R. Sundrum. “Out of this world supersymmetry breaking”. In: *Nucl. Phys. B* 557 (1999), pp. 79–118. DOI: 10.1016/S0550-3213(99)00359-4. arXiv: hep-th/9810155.

[79] H. Fukuda et al. “Higgsino Dark Matter or Not: Role of Disappearing Track Searches at the LHC and Future Colliders”. In: *Phys. Lett. B* 781 (2018), pp. 306–311. DOI: 10.1016/j.physletb.2018.03.088. arXiv: 1703.09675 [hep-ph].

[80] M. Papucci, J. T. Ruderman, and A. Weiler. “Natural SUSY Endures”. In: *JHEP* 09 (2012), p. 035. DOI: 10.1007/JHEP09(2012)035. arXiv: 1110.6926 [hep-ph].

[81] LUX Collaboration. “Results from a search for dark matter in the complete LUX exposure”. In: *Phys. Rev. Lett.* 118.2 (2017), p. 021303. DOI: 10.1103/PhysRevLett.118.021303. arXiv: 1608.07648 [astro-ph.CO].

[82] S. Thomas and J. D. Wells. “Phenomenology of Massive Vectorlike Doublet Leptons”. In: *Phys. Rev. Lett.* 81 (1 July 1998), pp. 34–37. DOI: 10.1103/PhysRevLett.81.34. URL: https://link.aps.org/doi/10.1103/PhysRevLett.81.34.

[83] ATLAS Collaboration. “The ATLAS Experiment at the CERN Large Hadron Collider”. In: *JINST* 3 (2008), S08003. DOI: 10.1088/1748-0221/3/08/S08003.

[84] L. Evans and P. Bryant. “LHC Machine”. In: *Journal of Instrumentation* 3.08 (Aug. 2008), S08001–S08001. DOI: 10.1088/1748-0221/3/08/s08001.

[85] CMS Collaboration. “The CMS experiment at the CERN LHC”. In: *Journal of Instrumentation* 3.08 (Aug. 2008), S08004–S08004. DOI: 10.1088/1748-0221/3/08/s08004.

[86] LHCb Collaboration. “The LHCb Detector at the LHC”. In: *Journal of Instrumentation* 3.08 (Aug. 2008), S08005–S08005. DOI: 10.1088/1748-0221/3/08/s08005.

[87] ALICE Collaboration. “The ALICE experiment at the CERN LHC”. In: *Journal of Instrumentation* 3.08 (Aug. 2008), S08002–S08002. DOI: 10.1088/1748-0221/3/08/s08002.

[88] O. C. Endner. “Search for New Physics Using Jets in Proton-Proton Collisions”. Presented 19 May 2016. PhD thesis. Mainz U., 2015. URL: https://cds.cern.ch/record/2154631.

[89] W. Herr and B. Muratori. “Concept of luminosity”. In: (2006). DOI: 10.5170/CERN-2006-002.361. URL: https://cds.cern.ch/record/941318.

[90] J. Pequenao. *Computer generated image of the whole ATLAS detector*. 2008. URL: https://cds.cern.ch/record/1095924.

[91] ATLAS Collaboration. *ATLAS Insertable B-Layer Technical Design Report*. Tech. rep. CERN-LHCC-2010-013. ATLAS-TDR-19. Sept. 2010. URL: https://cds.cern.ch/record/1291633.

[92] ATLAS Collaboration. “Performance of the ATLAS Track Reconstruction Algorithms in Dense Environments in LHC Run 2”. In: *Eur. Phys. J. C* 77.10 (2017), p. 673. DOI: 10.1140/epjc/s10052-017-5225-7. arXiv: 1704.07983 [hep-ex].

[93] ATLAS Collaboration. *ATLAS pixel detector: Technical Design Report*. Tech. rep. Geneva, 1998. URL: https://cds.cern.ch/record/381263.

[94] ATLAS Collaboration. *ATLAS Inner Detector: Technical Design Report, 1*. Tech. rep. CERN-LHCC-97-016. Geneva: CERN, July 1997. URL: https://cds.cern.ch/record/331063.

[95] ATLAS Collaboration. *ATLAS inner detector: Technical Design Report, 2*. Tech. rep. Geneva, 1997. URL: https://cds.cern.ch/record/331064.

[96] ATLAS Collaboration. *IBL Efficiency and Single Point Resolution in Collision Events*. Tech. rep. ATL-INDET-PUB-2016-001. Geneva: CERN, Aug. 2016. URL: https://cds.cern.ch/record/2203893.

[97] ATLAS TRT Collaboration. "The ATLAS Transition Radiation Tracker (TRT) proportional drift tube: Design and performance". In: *JINST* 3 (2008), P02013. DOI: 10.1088/1748-0221/3/02/P02013.

[98] ATLAS Collaboration. "Electron and photon energy calibration with the ATLAS detector using LHC Run 1 data". In: *Eur. Phys. J. C* 74.10 (2014), p. 3071. DOI: 10.1140/epjc/s10052-014-3071-4. arXiv: 1407.5063 [hep-ex].

[99] ATLAS Collaboration. *ATLAS liquid-argon calorimeter: Technical Design Report*. Tech. rep. Geneva: CERN, 1996. URL: https://cds.cern.ch/record/331061.

[100] ATLAS Collaboration. *ATLAS tile calorimeter: Technical Design Report*. Tech. rep. Geneva: CERN, 1996. URL: https://cds.cern.ch/record/331062.

[101] ATLAS Liquid Argon HEC Collaboration. "Performance of the ATLAS hadronic end-cap calorimeter in beam tests". In: *Nucl. Instrum. Meth. A* 482 (2002), pp. 94–124. DOI: 10.1016/S0168-9002(01)01338-9.

[102] ATLAS Collaboration. *ATLAS muon spectrometer: Technical Design Report*. Tech. rep. Geneva, 1997. URL: https://cds.cern.ch/record/331068.

[103] T. Kawamoto et al. *New Small Wheel Technical Design Report*. Tech. rep. CERN-LHCC-2013-006; ATLAS-TDR-020. ATLAS New Small Wheel Technical Design Report. June 2013. URL: https://cds.cern.ch/record/1552862.

[104] ATLAS Collaboration. *ATLAS central solenoid: Technical Design Report*. Tech. rep. Geneva: CERN, 1997. URL: https://cds.cern.ch/record/331067.

[105] ATLAS Collaboration. *ATLAS barrel toroid: Technical Design Report*. Tech. rep. Geneva: CERN, 1997. URL: https://cds.cern.ch/record/331065.

[106] ATLAS Collaboration. *ATLAS end-cap toroids: Technical Design Report*. Tech. rep. Geneva: CERN, 1997. URL: https://cds.cern.ch/record/331066.

[107] J. Hrdinka. "Tracker performance studies at the 100 TeV future circular hadron collider at extreme pile-up conditions". Presented 20 Sep 2019. PhD thesis. Vienna, Tech. U, 2019. URL: http://cds.cern.ch/record/2690632.

[108] M. Elsing et al. "The ATLAS Tier-0: Overview and operational experience". In: *J. Phys. Conf. Ser.* 219 (2010). Ed. by J. Gruntorad and M. Lokajicek, p. 072011. DOI: 10.1088/1742-6596/219/7/072011.

[109] *Athena*. Version 22.0.1. Apr. 2019. DOI: 10.5281/zenodo.2641996.

[110] G. Barrand et al. "GAUDI - A software architecture and framework for building HEP data processing applications". In: *Comput. Phys. Commun.* 140 (2001), pp. 45–55. DOI: 10.1016/S0010-4655(01)00254-5.

[111] E. Moyse and F. Åkesson. "The ATLAS Event Data Model". CHEP 2006 (Mumbai). Feb. 14, 2006. URL: https://indico.cern.ch/event/408139/contributions/979862/.

[112] A. Buckley et al. *Implementation of the ATLAS Run 2 event data model*. Tech. rep. ATL-SOFT-PROC-2015-003. 7. Geneva: CERN, May 2015. DOI: 10.1088/1742-6596/664/7/072045. URL: https://cds.cern.ch/record/2014150.

[113] J. Catmore et al. *A New Petabyte-scale Data Derivation Framework for ATLAS*. Tech. rep. ATL-SOFT-PROC-2015-041. 7. Geneva: CERN, May 2015. DOI: 10.1088/1742-6596/664/7/072007. URL: https://cds.cern.ch/record/2016628.

[114] ATLAS Collaboration. "Performance of electron and photon triggers in ATLAS during LHC Run 2". In: *Eur. Phys. J. C* 80.1 (2020), p. 47. DOI: 10.1140/epjc/s10052-019-7500-2. arXiv: 1909.00761 [hep-ex].

[115] ATLAS Collaboration. *The ATLAS Tau Trigger in Run 2*. Tech. rep. ATLAS-CONF-2017-061. CERN, July 2017. URL: https://cds.cern.ch/record/2274201.

[116] ATLAS Collaboration. *ATLAS Level-1 Calorimeter Trigger Algorithms*. Tech. rep. Geneva: CERN, Sept. 2004. URL: http://cds.cern.ch/record/792528.

[117] ATLAS Collaboration. "Performance of the ATLAS muon triggers in Run 2". In: *JINST* 15.09 (2020), P09015. DOI: 10.1088/1748-0221/15/09/p09015. arXiv: 2004.13447 [hep-ex].

[118] ATLAS Collaboration. "Operation of the ATLAS trigger system in Run 2". In: *JINST* 15.10 (2020), P10004. DOI: 10.1088/1748-0221/15/10/P10004. arXiv: 2007.12539 [physics.ins-det].

[119] ATLAS Collaboration. "Performance of the missing transverse momentum triggers for the ATLAS detector during Run-2 data taking". In: *JHEP* 08 (2020), p. 080. DOI: 10.1007/JHEP08(2020)080. arXiv: 2005.09554 [hep-ex].

[120] ATLAS Collaboration. *The ATLAS Data Acquisition system in LHC Run 2*. Tech. rep. ATL-DAQ-PROC-2017-007. 3. Geneva: CERN, Feb. 2017. DOI: 10.1088/1742-6596/898/3/032017. URL: https://cds.cern.ch/record/2244345.

[121] ATLAS Collaboration. "Topological cell clustering in the ATLAS calorimeters and its performance in LHC Run 1". In: *Eur. Phys. J. C* 77 (2017), p. 490. DOI: 10.1140/epjc/s10052-017-5004-5. arXiv: 1603.02934 [hep-ex].

[122] ATLAS Collaboration. *ATLAS Jet Reconstruction, Calibration, and Tagging of Lorentz-boosted Objects*. Tech. rep. ATL-PHYS-PROC-2017-236. Geneva: CERN, Nov. 2017. URL: https://cds.cern.ch/record/2291608.

[123] ATLAS Collaboration. "Electron and photon performance measurements with the ATLAS detector using the 2015–2017 LHC proton-proton collision data". In: *JINST* 14.12 (2019), P12006. DOI: 10.1088/1748-0221/14/12/P12006. arXiv: 1908.00005 [hep-ex].

[124] ATLAS Collaboration. "Muon reconstruction performance of the ATLAS detector in proton–proton collision data at $\sqrt{s}$ =13 TeV". In: *Eur. Phys. J. C* 76.5 (2016), p. 292. DOI: 10.1140/epjc/s10052-016-4120-y. arXiv: 1603.05598 [hep-ex].

[125] M. Cacciari, G. P. Salam, and G. Soyez. "The anti-k_t jet clustering algorithm". In: *JHEP* 04 (2008), p. 063. DOI: 10.1088/1126-6708/2008/04/063. arXiv: 0802.1189 [hep-ph].

[126] ATLAS Collaboration. "Jet reconstruction and performance using particle flow with the ATLAS Detector". In: *Eur. Phys. J. C* 77.7 (2017), p. 466. DOI: 10.1140/epjc/s10052-017-5031-2. arXiv: 1703.10485 [hep-ex].

[127] ATLAS Collaboration. *Muon reconstruction and identification efficiency in ATLAS using the full Run 2 pp collision data set at $\sqrt{s} = 13$ TeV*. Dec. 2020. arXiv: 2012.00578 [hep-ex].

[128] ATLAS Collaboration. "Performance of missing transverse momentum reconstruction with the ATLAS detector using proton-proton collisions at $\sqrt{s}$ = 13 TeV". In: *Eur. Phys. J. C* 78.11 (2018), p. 903. DOI: 10.1140/epjc/s10052-018-6288-9. arXiv: 1802.08168 [hep-ex].

[129] J. Alwall et al. "The automated computation of tree-level and next-to-leading order differential cross sections, and their matching to parton shower simulations". In: *JHEP* 07 (2014), p. 079. DOI: 10.1007/JHEP07(2014)079. arXiv: 1405.0301 [hep-ph].

[130] T. Sjöstrand et al. "An Introduction to PYTHIA 8.2". In: *Comput. Phys. Commun.* 191 (2015), pp. 159–177. DOI: 10.1016/j.cpc.2015.01.024. arXiv: 1410.3012 [hep-ph].

[131] D. J. Lange. "The EvtGen particle decay simulation package". In: *Nucl. Instrum. Meth. A* 462 (2001). Ed. by S. Erhan, P. Schlein, and Y. Rozen, pp. 152–155. DOI: 10.1016/S0168-9002(01)00089-4.

[132] GEANT4 Collaboration. “GEANT4: A Simulation toolkit”. In: *Nucl. Instrum. Meth. A* 506 (2003), pp. 250–303. DOI: 10.1016/S0168-9002(03)01368-8.

[133] ATLAS Collaboration. “Jet energy scale and resolution measured in proton-proton collisions at $\sqrt{s} = 13$ TeV with the ATLAS detector”. In: (July 2020). arXiv: 2007.02645 [hep-ex]. URL: https://cds.cern.ch/record/2722869.

[134] *Number of Interactions per Crossing*. https://twiki.cern.ch/twiki/bin/view/AtlasPublic/LuminosityPublicResultsRun2#Pileup_Interactions_and_Data_Tak. 2018.

[135] ATLAS Collaboration. “Luminosity determination in pp collisions at $\sqrt{s}$ = 8 TeV using the ATLAS detector at the LHC”. In: *Eur. Phys. J. C* 76.12 (2016), p. 653. DOI: 10.1140/epjc/s10052-016-4466-1. arXiv: 1608.03953 [hep-ex].

[136] G. Avoni et al. “The new LUCID-2 detector for luminosity measurement and monitoring in ATLAS”. In: *JINST* 13.07 (2018), P07017. DOI: 10.1088/1748-0221/13/07/P07017.

[137] ATLAS Collaboration. *Luminosity determination in pp collisions at $\sqrt{s} = 13$ TeV using the ATLAS detector at the LHC*. Tech. rep. Geneva: CERN, June 2019.

[138] T. Cornelissen et al. “The new ATLAS track reconstruction (NEWT)”. In: *J. Phys. Conf. Ser.* 119 (2008), p. 032014. DOI: 10.1088/1742-6596/119/3/032014.

[139] G. Charpak et al. “The Use of Multiwire Proportional Counters to Select and Localize Charged Particles”. In: *Nucl. Instrum. Meth.* 62 (1968), pp. 262–268. DOI: 10.1016/0029-554X(68)90371-6.

[140] P. F. Åkesson et al. *ATLAS Tracking Event Data Model*. Tech. rep. ATL-SOFT-PUB-2006-004. ATL-COM-SOFT-2006-005. CERN-ATL-COM-SOFT-2006-005. Geneva: CERN, July 2006. URL: http://cds.cern.ch/record/973401.

[141] C. Runge. “Über die numerische Auflösung von Differentialgleichungen”. In: *Mathematische Annalen* 46 (1895), pp. 167–178. DOI: 10.1007/BF01446807.

[142] W. Kutta. “Beitrag zur näherungweisen Integration totaler Differentialgleichungen”. In: *Zeitschrift für Mathematik und Physik* 46 (1901), pp. 435–453.

[143] E. J. Nyström. “Über Die Praktische Auflösung von Integralgleichungen mit Anwendungen auf Randwertaufgaben”. In: *Acta Math.* 54 (1930), pp. 185–204. DOI: 10.1007/BF02547521.

[144] J. Myrheim and L. Bugge. “A fast Runge-Kutta method for fitting tracks in a magnetic field”. In: *Nucl. Instrum. Meth.* 160.1 (1979), pp. 43–48. DOI: 10.1016/0029-554X(79)90163-0.

[145] L. Bugge and J. Myrheim. “Tracking and track fitting”. In: *Nucl. Instrum. Meth.* 179 (1981), pp. 365–381. DOI: 10.1016/0029-554X(81)90063-X.

[146] E. Lund et al. “Track parameter propagation through the application of a new adaptive Runge-Kutta-Nystroem method in the ATLAS experiment”. In: *JINST* 4 (2009), P04001. DOI: 10.1088/1748-0221/4/04/P04001.

[147] A. Strandlie and R. Frühwirth. “Track and vertex reconstruction: From classical to adaptive methods”. In: *Rev. Mod. Phys.* 82 (2010), pp. 1419–1458. DOI: 10.1103/RevModPhys.82.1419.

[148] R. Frühwirth and A. Strandlie. “Track fitting with ambiguities and noise: A study of elastic tracking and nonlinear filters”. In: *Comput. Phys. Commun.* 120.2-3 (1999), pp. 197–214. DOI: 10.1016/S0010-4655(99)00231-3.

[149] R. Frühwirth. “Track fitting with non Gaussian noise”. In: *Comput. Phys. Commun.* 100 (1997), pp. 1–16. DOI: 10.1016/S0010-4655(96)00155-5.

[150] R. Frühwirth and S. Frühwirth-Schnatter. “On the treatment of energy loss in track fitting”. In: *Comput. Phys. Commun.* 110.1-3 (1998), pp. 80–86. DOI: 10.1016/S0010-4655(97)00157-4.

[151] E. Lund et al. “Transport of covariance matrices in the inhomogeneous magnetic field of the ATLAS experiment by the application of a semi-analytical method”. In: *JINST* 4 (2009), P04016. DOI: 10.1088/1748-0221/4/04/P04016.

[152] E. Lund et al. *Treatment of energy loss and multiple scattering in the context of track parameter and covariance matrix propagation in continuous material in the ATLAS experiment*. Tech. rep. CERN, July 2008. URL: https://cds.cern.ch/record/1114577.

[153] A. Salzburger, Š. Todorova, and M. W. Wolter. *The ATLAS Tracking Geometry Description*. Tech. rep. ATL-SOFT-PUB-2007-004. ATL-COM-SOFT-2007-009. Geneva: CERN, June 2007. URL: https://cds.cern.ch/record/1038098.

[154] A. Rosenfeld and J. L. Pfaltz. “Sequential Operations in Digital Picture Processing”. In: *J. ACM* 13.4 (Oct. 1966), pp. 471–494. DOI: 10.1145/321356.321357.

[155] ATLAS Collaboration. “A neural network clustering algorithm for the ATLAS silicon pixel detector”. In: *JINST* 9 (2014), P09009. DOI: 10.1088/1748-0221/9/09/P09009. arXiv: 1406.7690 [hep-ex].

[156] P. V. C. Hough. “Machine Analysis of Bubble Chamber Pictures”. In: *Conf. Proc. C* 590914 (1959). Ed. by L. Kowarski, pp. 554–558.

[157] F. Pantaleo. “New Track Seeding Techniques for the CMS Experiment”. PhD thesis. Hamburg U., 2017. URL: https://cds.cern.ch/record/2293435.

[158] ATLAS Collaboration. *Performance of the ATLAS Silicon Pattern Recognition Algorithm in Data and Simulation at $\sqrt{s} = 7$ TeV*. Tech. rep. ATLAS-CONF-2010-072. Geneva: CERN, July 2010. URL: https://cds.cern.ch/record/1281363.

[159] R. E. Kalman. “A New Approach to Linear Filtering and Prediction Problems”. In: *Journal of Basic Engineering* 82.1 (Mar. 1960), pp. 35–45. DOI: 10.1115/1.3662552.

[160] R. Frühwirth. “Application of Kalman filtering to track and vertex fitting”. In: *Nucl. Instrum. Meth. A* 262.2-3 (1987), pp. 444–450. DOI: 10.1016/0168-9002(87)90887-4.

[161] T. Cornelissen. “The global chi2 track fitter in ATLAS”. CHEP 2007 (Victoria). Sept. 5, 2007. URL: https://indico.cern.ch/event/3580/contributions/1768469/.

[162] S. Fleischmann. “Track reconstruction in the ATLAS experiment: The deterministic annealing filter”. PhD thesis. Wuppertal U., 2006.

[163] ATLAS Collaboration. *Development of ATLAS Primary Vertex Reconstruction for LHC Run 3*. Tech. rep. ATL-PHYS-PUB-2019-015. Geneva: CERN, Apr. 2019. URL: https://cds.cern.ch/record/2670380.

[164] P. Billoir and S. Qian. “Fast vertex fitting with a local parametrization of tracks”. In: *Nucl. Instrum. Meth. A* 311.1-2 (1992), pp. 139–150. DOI: 10.1016/0168-9002(92)90859-3.

[165] W. Waltenberger, R. Frühwirth, and P. Vanlaer. “Adaptive vertex fitting”. In: *J. Phys. G* 34 (2007), N343. DOI: 10.1088/0954-3899/34/12/N01.

[166] ATLAS Collaboration. *Study of the mechanical stability of the ATLAS Insertable B-Layer*. Tech. rep. ATL-INDET-PUB-2015-001. Geneva: CERN, June 2015. URL: https://cds.cern.ch/record/2022587.

[167] ATLAS Collaboration. *Alignment of the ATLAS Inner Detector in Run-2*. Tech. rep. CERN-EP-2020-108. Geneva: CERN, July 2020. arXiv: 2007.07624 [hep-ex]. URL: https://cds.cern.ch/record/2724037.

[168] P. Calafiura et al. *ATLAS HL-LHC Computing Conceptual Design Report*. Tech. rep. CERN-LHCC-2020-015. LHCC-G-178. Geneva: CERN, Sept. 2020. URL: https://cds.cern.ch/record/2729668.

[169] G. Apollinari et al. *High-Luminosity Large Hadron Collider (HL-LHC): Preliminary Design Report*. CERN Yellow Reports: Monographs. Geneva: CERN, 2015. DOI: 10.5170/CERN-2015-005. URL: https://cds.cern.ch/record/2116337.

[170] ATLAS Collaboration. *Technical Design Report: A High-Granularity Timing Detector for the ATLAS Phase-II Upgrade*. Tech. rep. CERN-LHCC-2020-007; ATLAS-TDR-031. Geneva: CERN, June 2020. URL: https://cds.cern.ch/record/2719855.

[171] C. Leggett et al. *AthenaMT: Upgrading the ATLAS Software Framework for the Many-Core World with Multi-Threading*. Oct. 2016. URL: https://cds.cern.ch/record/2222298.

[172] ATLAS Collaboration. *Technical Design Report for the ATLAS Inner Tracker Strip Detector*. Tech. rep. CERN-LHCC-2017-005; ATLAS-TDR-025. Geneva: CERN, Apr. 2017. URL: https://cds.cern.ch/record/2257755.

[173] ATLAS Collaboration. *Technical Design Report for the ATLAS Inner Tracker Pixel Detector*. Tech. rep. CERN-LHCC-2017-021; ATLAS-TDR-030. Geneva: CERN, Sept. 2017. URL: http://cds.cern.ch/record/2285585.

[174] ATLAS Collaboration. *Expected Tracking Performance of the ATLAS Inner Tracker at the HL-LHC*. Tech. rep. ATL-PHYS-PUB-2019-014. Geneva: CERN, Mar. 2019. URL: https://cds.cern.ch/record/2669540.

[175] *ITk Pixel Layout Updates*. https://atlas.web.cern.ch/Atlas/GROUPS/PHYSICS/PLOTS/ITK-2020-002/. Apr. 2020.

[176] *HEPiX Benchmarking Working Group*. http://w3.hepix.org/benchmarking.html.

[177] ATLAS Collaboration. *Fast Track Reconstruction for HL-LHC*. Tech. rep. ATL-PHYS-PUB-2019-041. Geneva: CERN, Oct. 2019. URL: https://cds.cern.ch/record/2693670.

[178] ATLAS Collaboration. *Technical Design Report for the Phase-II Upgrade of the ATLAS TDAQ System*. Tech. rep. CERN-LHCC-2017-020; ATLAS-TDR-029. Geneva: CERN, Sept. 2017. URL: https://cds.cern.ch/record/2285584.

[179] A. Salzburger et al. *Acts Project*. Version v0.15.00. Jan. 2020. DOI: 10.5281/zenodo.3606011.

[180] *CLHEP*. https://proj-clhep.web.cern.ch/proj-clhep/.

[181] G. Guennebaud, B. Jacob, et al. *Eigen v3*. http://eigen.tuxfamily.org. 2010.

[182] S. Amrouche et al. "The Tracking Machine Learning challenge : Accuracy phase". In: (Apr. 2019). DOI: 10.1007/978-3-030-29135-8_9. arXiv: 1904.06778 [hep-ex].

[183] *TensorFlow*. https://www.tensorflow.org/.

[184] *KDE*. https://kde.org/.

[185] *Boost*. https://www.boost.org.

[186] *Threading Building Blocks*. https://github.com/oneapi-src/oneTBB.

[187] R. Brun and F. Rademakers. "ROOT: An object oriented data analysis framework". In: *Nucl. Instrum. Meth. A* 389 (1997). Ed. by M. Werlen and D. Perret-Gallix, pp. 81–86. DOI: 10.1016/S0168-9002(97)00048-X.

[188] M. Frank et al. *AIDASoft/DD4hep*. Oct. 2018. DOI: 10.5281/zenodo.592244. URL: http://dd4hep.cern.ch/.

[189] N. Lohmann et al. *nlohmann::json*. https://github.com/nlohmann/json.

[190] C. Arnault. *Configuration Management Tool*. http://www.cmtsite.net.

[191] *Subversion*. https://subversion.apache.org.

[192] *Git*. https://git-scm.com/.

[193] *CMake*. https://cmake.org.

[194] *GitHub*. https://github.com.

[195] *GitLab*. https://about.gitlab.com.

[196] *Gitea*. https://gitea.com.

[197] *GitLab at CERN*. https://gitlab.cern.ch.

[198] *Acts Project on GitHub*. https://github.com/acts-project/acts.

[199] *Scientific Linux CERN*. https://linux.web.cern.ch/scientific6/.

[200] *CERN CentOS 7*. https://linux.web.cern.ch/centos7/.

[201] *Worldwide LHC Computing Grid*. https://wlcg-public.web.cern.ch/.

[202] Belle-II Collaboration. *Belle II Technical Design Report*. Nov. 2010. arXiv: 1011.0352 [physics.ins-det].

[203] CEPC Study Group Collaboration. *CEPC Conceptual Design Report: Volume 2 - Physics & Detector*. Ed. by J. B. Guimarães da Costa et al. Nov. 2018. arXiv: 1811.10545 [hep-ex].

[204] FASER Collaboration. *Letter of Intent for FASER: ForwArd Search ExpeRiment at the LHC*. Tech. rep. CERN-LHCC-2018-030. Geneva: CERN, Nov. 2018. arXiv: 1811.10243 [physics.ins-det].

[205] PHENIX Collaboration. *An Upgrade Proposal from the PHENIX Collaboration*. Jan. 2015. arXiv: 1501.06197 [nucl-ex].

[206] *LAPACK*. https://github.com/Reference-LAPACK/lapack.

[207] V. Tsulaia and J. Boudreau. "The GeoModel Toolkit for Detector Description". https://indico.cern.ch/event/0/contributions/1294152/. CHEP 2004 (Interlaken). Sept. 2004.

[208] ATLAS Collaboration. "The ATLAS Simulation Infrastructure". In: *Eur. Phys. J. C* 70 (2010), pp. 823–874. DOI: 10.1140/epjc/s10052-010-1429-9. arXiv: 1005.4568 [physics.ins-det].

[209] Wavefront Technologies. *Wavefront OBJ format*. http://www.martinreddy.net/gfx/3d/OBJ.spec.

[210] N. Hessey. *Building a Stereo-angle into strip-sensors for the ATLAS-Upgrade Inner-Tracker Endcaps*. Tech. rep. ATL-UPGRADE-PUB-2013-002. Geneva: CERN, Feb. 2013. URL: https://cds.cern.ch/record/1514636.

[211] P. C. Mahalanobis. "On the generalized distance in statistics". In: *Proceedings of the National Institute of Sciences (Calcutta)* 2 (1936), pp. 49–55.

[212] *Shapely*. https://github.com/Toblerity/Shapely.

[213] A. Woo. "Fast Ray-Box Intersection". In: *Graphics Gems*. Ed. by A. S. Glassner. San Diego: Morgan Kaufmann, 1990, pp. 395–396. ISBN: 978-0-08-050753-8. DOI: https://doi.org/10.1016/B978-0-08-050753-8.50084-X.

[214] T. Barnes. *Fast, branchless Ray/Bounding Box intersections*. Accessed 2020-11-05. May 2011. URL: https://tavianator.com/fast-branchless-raybounding-box-intersections/.

[215] T. Barnes. *Fast, branchless Ray/Bounding Box intersections, Part 2: NaNs*. Accessed 2020-11-05. Mar. 2015. URL: https://tavianator.com/fast-branchless-raybounding-box-intersections-part-2-nans/.

[216] U. Assarsson and T. Moller. "Optimized View Frustum Culling Algorithms for Bounding Boxes". In: *Journal of Graphics Tools* 5.1 (2000), pp. 9–22. DOI: 10.1080/10867651.2000.10487517.

[217] ATLAS Collaboration. *Performance of tracking and vertexing techniques for a disappearing track plus soft track signature with the ATLAS detector*. Tech. rep. ATL-PHYS-PUB-2019-011. Geneva: CERN, Mar. 2019. URL: https://cds.cern.ch/record/2669015.

[218] ATLAS Collaboration. "Search for charginos nearly mass degenerate with the lightest neutralino based on a disappearing-track signature in pp collisions at $\sqrt{(s)}$=8 TeV with the ATLAS detector". In: *Phys. Rev. D* 88.11 (2013), p. 112006. DOI: 10.1103/PhysRevD.88.112006. arXiv: 1310.3675 [hep-ex].

[219] ATLAS Collaboration. "Search for long-lived charginos based on a disappearing-track signature in pp collisions at $\sqrt{s} = 13$ TeV with the ATLAS detector". In: *JHEP* 2018.6 (2018), p. 022. DOI: 10.1007/JHEP06(2018)022. arXiv: 1712.02118 [hep-ex].

[220] M. Battaglia and S. Miglioranzi. *A Study of Tracklet Reconstruction with the ATLAS Pixel Detector*. Tech. rep. ATL-COM-PHYS-2016-1146 (ATLAS Internal). Geneva: CERN, Aug. 2016. URL: https://cds.cern.ch/record/2208731.

[221] N. Nagata and S. Shirai. "Higgsino Dark Matter in High-Scale Supersymmetry". In: *JHEP* 01 (2015), p. 029. DOI: 10.1007/JHEP01(2015)029. arXiv: 1410.4549 [hep-ph].

[222] CMS Collaboration. "Search for disappearing tracks in proton-proton collisions at $\sqrt{s} = 13$ TeV". In: *Phys. Lett. B* 806 (2020), p. 135502. DOI: 10.1016/j.physletb.2020.135502. arXiv: 2004.05153 [hep-ex].

[223] ATLAS Collaboration. "Search for direct chargino production in anomaly-mediated supersymmetry breaking models based on a disappearing-track signature in pp collisions at $\sqrt{s} = 7$ TeV with the ATLAS detector". In: *JHEP* 01 (2013), p. 131. DOI: 10.1007/JHEP01(2013)131. arXiv: 1210.2852 [hep-ex].

[224] CMS Collaboration. "Search for disappearing tracks in proton-proton collisions at $\sqrt{s} = 8$ TeV". In: *JHEP* 01 (2015), p. 096. DOI: 10.1007/JHEP01(2015)096. arXiv: 1411.6006 [hep-ex].

[225] CMS Collaboration. "Search for disappearing tracks as a signature of new long-lived particles in proton-proton collisions at $\sqrt{s} = 13$ TeV". In: *JHEP* 08 (2018), p. 016. DOI: 10.1007/JHEP08(2018)016. arXiv: 1804.07321 [hep-ex].

[226] ATLAS Collaboration. "Search for metastable heavy charged particles with large ionization energy loss in pp collisions at $\sqrt{s} = 13$ TeV using the ATLAS experiment". In: *Phys. Rev. D* 93.11 (2016), p. 112015. DOI: 10.1103/PhysRevD.93.112015. arXiv: 1604.04520 [hep-ex].

[227] ATLAS Collaboration. "Search for heavy long-lived charged R-hadrons with the ATLAS detector in 3.2 fb^{-1} of proton–proton collision data at $\sqrt{s} = 13$ TeV". In: *Phys. Lett. B* 760 (2016), pp. 647–665. DOI: 10.1016/j.physletb.2016.07.042. arXiv: 1606.05129 [hep-ex].

[228] CMS Collaboration. "Search for new physics in events with two soft oppositely charged leptons and missing transverse momentum in proton-proton collisions at $\sqrt{s} = 13$ TeV". In: *Phys. Lett. B* 782 (2018), pp. 440–467. DOI: 10.1016/j.physletb.2018.05.062. arXiv: 1801.01846 [hep-ex].

[229] ATLAS Collaboration. *Search for direct pair production of higgsinos by the reinterpretation of the disappearing track analysis with 36.1 fb^{-1} of $\sqrt{s} = 13$ TeV data collected with the ATLAS experiment*. Tech. rep. ATL-PHYS-PUB-2017-019. Geneva: CERN, Dec. 2017. URL: https://cds.cern.ch/record/2297480.

[230] *Data reprocessing for VH(4b) long-lived search (ATLAS Internal)*. https://its.cern.ch/jira/browse/DATREP-156. Accessed 2020-11-05.

[231] *Total Integrated Luminosity and Data Quality in 2018*. https://twiki.cern.ch/twiki/bin/view/AtlasPublic/LuminosityPublicResultsRun2. 2018.

[232] F. E. Paige et al. *ISAJET 7.69: A Monte Carlo event generator for pp, anti-p p, and e+e- reactions*. Dec. 2003. arXiv: hep-ph/0312045.

[233] ATLAS Collaboration. *ATLAS Pythia 8 tunes to 7 TeV data*. Tech. rep. ATL-PHYS-PUB-2014-021. Geneva: CERN, Nov. 2014. URL: https://cds.cern.ch/record/1966419.

[234] R. D. Ball et al. "Parton distributions with LHC data". In: *Nucl. Phys. B* 867 (2013), pp. 244–289. DOI: 10.1016/j.nuclphysb.2012.10.003. arXiv: 1207.1303 [hep-ph].

[235] D. Adams et al. *Recommendations of the Physics Objects and Analysis Harmonisation Study Groups 2014*. Tech. rep. ATL-PHYS-INT-2014-018 (ATLAS Internal). Geneva: CERN, July 2014. URL: https://cds.cern.ch/record/1743654.

[236] ATLAS Collaboration. *Selection of jets produced in 13TeV proton-proton collisions with the ATLAS detector*. Tech. rep. ATLAS-CONF-2015-029. Geneva: CERN, July 2015. URL: https://cds.cern.ch/record/2037702.

[237] ATLAS Collaboration. *Search for degenerate chargino production in pp collisions at $\sqrt{s}$ = 13 TeV with a disappearing track signature with the ATLAS detector*. Tech. rep. ATL-COM-PHYS-2019-494 (ATLAS Internal). Geneva: CERN, May 2019. URL: https://cds.cern.ch/record/2673616.

[238] G. Cowan et al. "Asymptotic formulae for likelihood-based tests of new physics". In: *Eur. Phys. J. C* 71 (2011). [Erratum: Eur.Phys.J.C 73, 2501 (2013)], p. 1554. DOI: 10.1140/epjc/s10052-011-1554-0. arXiv: 1007.1727 [physics.data-an].

[239] J. Neyman and E. S. Pearson. "On the Problem of the Most Efficient Tests of Statistical Hypotheses". In: *Phil. Trans. Roy. Soc. Lond. A* 231.694-706 (1933), pp. 289–337. DOI: 10.1098/rsta.1933.0009.

[240] A. L. Read. "Presentation of search results: The CL(s) technique". In: *J. Phys. G* 28 (2002). Ed. by M. Whalley and L. Lyons, pp. 2693–2704. DOI: 10.1088/0954-3899/28/10/313.

[241] K. Cranmer et al. *HistFactory: A tool for creating statistical models for use with RooFit and RooStats*. Tech. rep. CERN-OPEN-2012-016. New York: New York U., Jan. 2012. URL: https://cds.cern.ch/record/1456844.

[242] ATLAS Collaboration. *ATLAS sensitivity to winos and higgsinos with a highly compressed mass spectrum at the HL-LHC*. Tech. rep. Geneva: CERN, 2018. URL: https://cds.cern.ch/record/2647294.

[243] FCC Collaboration. "FCC Physics Opportunities: Future Circular Collider Conceptual Design Report Volume 1". In: *Eur. Phys. J. C* 79.6 (2019), p. 474. DOI: 10.1140/epjc/s10052-019-6904-3.

[244] FCC Collaboration. "FCC-ee: The Lepton Collider: Future Circular Collider Conceptual Design Report Volume 2". In: *Eur. Phys. J. ST* 228.2 (2019), pp. 261–623. DOI: 10.1140/epjst/e2019-900045-4.

[245] FCC Collaboration. "FCC-hh: The Hadron Collider: Future Circular Collider Conceptual Design Report Volume 3". In: *Eur. Phys. J. ST* 228.4 (2019), pp. 755–1107. DOI: 10.1140/epjst/e2019-900087-0.

[246] M. Saito et al. "Discovery reach for wino and higgsino dark matter with a disappearing track signature at a 100 TeV pp collider". In: *Eur. Phys. J. C* 79.6 (2019), p. 469. DOI: 10.1140/epjc/s10052-019-6974-2. arXiv: 1901.02987 [hep-ph].

[247] M. Dobbs and J. B. Hansen. "The HepMC C++ Monte Carlo event record for High Energy Physics". In: *Comput. Phys. Commun.* 134 (2001), pp. 41–46. DOI: 10.1016/S0010-4655(00)00189-2.