IEKP-KA/2013-06

Karlsruhe Institute of Technology

A Local Tracking Algorithm for the Central Drift Chamber of Belle II

Oliver Frost

Diploma thesis

at the Department of Physics
at the Institute of Experimental Nuclear Physics (KIT)

Advisor: Prof. Dr. M. Feindt
Institute of Experimental Nuclear Physics

Coadvisor: Prof. Dr. U. Husemann
Institute of Experimental Nuclear Physics

March 5, 2013

IEKP-KA/2013-06

Karlsruher Institut für Technologie

Lokale Spurfindung in der zentralen Driftkammer des Belle II Detektors

Oliver Frost

Diplomarbeit

an der Fakultät für Physik
am Institut für Experimentelle Kernphysik (KIT)

Referent: Prof. Dr. M. Feindt
Institut für Experimentelle Kernphysik

Korreferent: Prof. Dr. U. Husemann
Institut für Experimentelle Kernphysik

5. März 2013

Ich versichere wahrheitsgemäß, die Arbeit selbstständig angefertigt, alle benutzten Hilfsmittel vollständig und genau angegeben und alles kenntlich gemacht zu haben, was aus Arbeiten anderer unverändert oder mit Abänderungen entnommen wurde.

Karlsruhe, 5. März 2013

. . . . . .

(**Oliver Frost**)

Zusammenfassung

Mit dem geplanten Belle-II-Experiment strebt das japanische Zentrum für Hochenergiephysik KEK die Untersuchung seltener Zerfälle im Bereich der sogenannten B-Physik mit bisher unerreichter Präzision an. Die erhöhte Genauigkeit soll dabei nicht nur durch die technologische Erneuerung des Detektors gegenüber seines Vorgängers und durch die auf das Vierzigfache gesteigerte Luminosität des Elektronen-Positronen-Speicherrings SuperKEKB verwirklicht werden, sondern auch durch die Verwendung neuer und verbesserter Methoden zur Rekonstruktion der physikalischen Informationen aus den Detektordaten.

Zur Unterstützung dieses Experiments würde am Institut für experimentelle Kernphysik am Karlsruher Institut für Technologie damit begonnen, einen leistungsfähigen Algorithmus zur Spurrekonstruktion für die zentrale Driftkammer des Belle-II-Detektors zu entwickeln. Der erhöhte Messuntergrund, der durch den intensiveren Stahl hervorgerufen wird, stellt dabei eine neue Herausforderungen dar, die frühzeitig in die Überlegungen zur Spurrekonstruktion einbezogen wird.

Die vorliegende Arbeit soll dazu einen grundlegenden Betrag leisten, indem sie fundamentale Prinzipien analysiert und sowohl ein Konzept für einen lokalen Spurfindungsalgorithmus implementiert als auch ein Gerüst für dessen detaillierte Evaluierung bereitstellt.

Der lokale Spurfindungsansatz

Im Allgemeinen kann man zwei fundamentale Ansätze zur Spurfindung unterscheiden, welche beide am Institut verfolgt werden. Der sogenannte globale Ansatz bezieht für die Mustererkennung die gesamte gemessene Ereignisstruktur ein und sucht nach Bahnkurven mit zuvor eingeschränkter Form.

Der lokale Ansatz konzentriert sich dagegen auf die direkten Beziehungen von örtlich benachbarten Messgrößen und baut Spuren stufenweise durch die Kombination von Teilstücken mit steigender Komplexität auf, wodurch der tatsächliche Verlauf von Teilchenbahnen mit einer größer Toleranz für Mehrfachstreuungen detailliert modelliert werden kann. Dies stellt einen wesentlichen Vorteil der lokalen Methode dar, da keine Beschränkung der Spurform vorgenommen werden muss. Die detaillierte Modellierung führt allerdings zu einer großen Anzahl frei einstellbarer Parameter oder ganzer Funktionsformen, die in einer gründlichen Untersuchung bestimmt werden müssen, bevor man

eine optimales Ergebnis erhalten kann. Darüber hinaus muss auf die optimal Handhabung der Daten in jeder Teilstufe geachtet werden, um einen schnellen Algorithmus zu erhalten, der Milliarden von Ereignissen in einer annehmbaren Zeit prozessieren kann.

Die konkrete Modellierung potentieller Teilchenbahnen besteht aus den folgenden zwei wesentlich Bestandteilen:

- Kombination von gemessen Größen zu Position von vermuteten Teilchen.
- Finden von mögliche Übergängen des Teilchens von einer Position zur nächsten.

Die beiden Bestandteile sind äquivalent zu Knoten und Kanten eines gerichteten Graphen, der den zeitlichen Verlauf der Teilchenbewegung nach den physikalischen Gesetzen möglichst akkurat abbilden soll. Die tatsächlichen Flugbahnen der Teilchen sind in diesem Graphen als Pfade kodiert. Um die hohe Symmetrie der physikalischen Gesetze widerzuspiegeln sollte weder eine Unterscheidung zwischen einwärts und auswärts noch zwischen im Uhrzeigersinn und gegen den Uhrzeigersinn in den Graphen einführen werden, sodass es auch möglich ist innerhalb des Detektors kreisende (engl. curling) Teilchen zu verfolgen.

Die Methode, mit der die wahrscheinlichsten Pfad aus diesem Graphen generiert werden, ist eine weitere entscheidende Wahl für die Qualität und die Geschwindigkeit des Algorithmus. Da ein vollständiges Backtracking des Graphen unter Einbeziehung aller einzelnen Verbindungen zu zeitaufwändig ist, wird für die Spurfindung häufig eine Form des Zellularautomaten eingesetzt, um lange Pfade zu erkennen, bevor sie konstruiert werden. Der klassische Zellularautomat für eine Spurfindungsanwendung legt dazu diskrete Wert in den einzelnen Knoten des Graphen ab, der die Reichweite des längsten Pfades ab dieser Position anzeigt. Der längste Pfad kann von der Zelle mit dem absolut höchsten Wert ausgehend durch die Verfolgung von hochwertigen Zellen erzeugt werden. Während dieser Arbeit konnte eine Verallgemeinerung dieses Prinzips abgeleitet werden, die in jedem Knoten und jeder Kante ein Gewicht zulässt und damit eine detailliertere Kodierung der Positions– und Verbindungsqualität erlaubt, womit ein Mittelweg zwischen dem klassischen Zellularautomaten und dem verwandten Denby–Peterson–Netzwerk eingeschlagen wird [18].

Aufbau der zentralen Driftkammer

Bevor der konkrete Entwurf des Algorithmus beschrieben werden kann, sei zunächst kurz das Messprinzip der zentralen Driftkammer des Belle–II–Experiments zusammengefasst.

Während der Passage eines geladenen Teilchens durch das Gas der Driftkammer werden durch dessen ionisierende Wirkung nach der Bethe-Bloch-Formel Elektronen aus der Hülle der Gasatome freigesetzt. Die Aufnahme der entstehenden Elektronen geschieht

in den elementaren Driftzellen, aus den die zentrale Driftkammer aufgebaut ist. Die Driftzellen werden durch acht lange parallel zur Strahlachse angeordnete Felddrähte begrenzt, die durch ihr großes abstoßendes Potential die Elektronen auf einen in der Mitte der Zelle befindlichen Messdraht beschleunigen und dadurch einen vorverstärkten messbaren Strompuls generieren. Die Driftzeit der Elektronen ist dabei nicht zu vernachlässigen und sie wird von der Ausleseelektronik aufgenommen, um den Abstand des ionisierenden Teilchens zum Messdraht zu bestimmen. Diese Driftlänge schränkt die Position der Ionisation auf einen Zylinder um den Messdraht ein und die zu rekonstruierende Teilchenbahn verläuft stets tangential zu diesem Zylinder. Während der Spurrekonstruktion können daher Tangenten an diesen Zylinder konstruiert werden, um eine linear Näherung der Teilchenbahn zu erhalten.

Des Weiteren unterscheidet man zwischen zwei Arten von Driftzellen. Sogenannte *Axialzellen* sind exakt parallel zur Strahlachse ausgerichtet und nehmen daher ein projiziertes Abbild der Teilchenbahnen entlang dieser Achse auf. Um auch in die dritte Dimension rekonstruieren zu können sind sogenannte *Stereozellen* vorhanden, die eine leichte Verdrehung um die Strahlachse aufweisen. Da die Rotation entlang der Strahlachse etwa linear zunimmt, ist sie ein Maß für die Tiefeninformation aus der Projektionsebene entlang der Strahlachse. Eine Stereozelle kann erst in Kombination mit einer zugeordneten Bahn aus der Projektionsebene in die dritte Dimension rekonstruiert werden da anders kein Referenzpunkt für die Rotation gegeben werden kann.

Axial– und Stereozellen sind in Lagen mit gleicher Entfernung zur Strahlachse angeordnet. Diese Lagen sind wiederum ineinandergeschachtelt in Superlagen von abwechselndem Axial– und Stereotype organisiert. Innerhalb einer Superlage zeigen die Driftzellen eine hexagonale Nahordnung, was es möglich macht eine Nachbarschaftsbeziehung zu konstruieren, die fundamental für die lokale Verfolgung von Spuren ist.

Des Weiteren ist die zentrale Driftkammer von einem Magnetfeld durchdrungen, sodass die geladene Zerfallsprodukte vom Interaktionspunkt der Teilchenkollisionen auf Helixbahnen gezwungen werden, deren Radius Auskunft über den Teilchenimpuls gibt.

Entwurf des Spurfindungsalgorithmus

Die Bestimmung diese Impulses stellt eins der Hauptziele der zentralen Driftkammer dar. Die Spurfindung ist bei dieser Messung dafür verantwortlich Driftzellen zu finden, die von dem selbem Teilchen entlang seiner Bahn getroffen wurden. Die endgültige Bestimmung der Bahnparameter findet jedoch nicht während dem hier vorgestellt Spurfindungsalgorithmus statt, sondern ist in einem separaten Schritt durch die Kalman-Filter-Implementierung von Genfit[1] realisiert. Jedoch können die Startparameter des Kalman-Filters aus der Spurfindung bestimmt werden.

Der Aufbau der zentralen Driftkammer des Belle–II-Experiments in Superlagen legt eine zweistufige Rekonstruktion der Teilchenbahnen nahe. Zuerst werden Spursegmen-

te innerhalb einer Superlage ausgehend von einzelnen aktivierten Driftzellen gebildet. Diese Segmente dienen dann in der zweiten Stufe als elementare Einheiten für den Aufbau vollständiger Spuren. Beide Stufen werden in etwa ähnlicher Weise abgehandelt, in dem zunächst Tripel der fundamentalen Einheiten gebildet werden, die sowohl Position also auch die momentane Bewegungsrichtung des vermuteten Teilchens darstellen. Die momentane Bewegungsrichtung kann im Folgenden dazu benutzt werden die Möglichkeit des Übergangs zu einem benachbarte Tripel zu entscheiden.

Segmentierung

In der ersten Stufe kann nur in der zweidimensionalen Projektionsebene entlang der Strahlachse rekonstruiert werden, da Segmente sich nur auf eine Superlage beschränken. Die Driftzylinder reduzieren sich in der Projektion zu Kreisen, welche ebenfalls tangential zu den projizierten Bahnen liegen. Die konkrete Prozedur verläuft zusammengefasst wie folgt:

1. Zunächst werden die aktivierten Driftzellen durch Expansion der lokalen Nachbarschaftsbeziehung zu lokalen Gruppen zusammengefasst.

2. In jeder Gruppe werden Tripel von benachbarten Driftzellen gebildet, die die Position des Teilchens triangulieren und als Knoten des zu konstruierenden Graphen dienen. Eine assoziierte Bewegungsrichtung kann durch die Konstruktion von Tangenten als lineare Näherung der Bahnkurve zwischen den Driftkreisen angenommen werden. Da auf der Größenskala von Driftzellen die Teilchenbahnen fasst linear sind, ist nur eine gering Winkelabweichung der Kombination von Tangenten zulässig und beschränkt daher die Anzahl der gültigen Tripel.

3. Ausgehend von der Bewegungsrichtung können nun potentielle Fortsetzungen von einem Tripel zum nächsten gesucht werden, die die Kanten des Graphen darstellen. Wieder ist nur eine geringe Winkelabweichung der Bewegungsrichtung in einem Tripel zum nächsten erlaubt, was die Zahl gültiger Kanten verringert.

4. Aus dem konstruiertem Graph können nun die Segmente durch eine wiederholte Anwendung des Zellulären Automaten gewonnen werden, wobei nach jeder Anwendung die verwendeten Knoten als genutzt markiert und im Folgenden ignoriert werden. Auf diese Weise ist es möglich mehrere disjunkte Segmente aus einer Gruppe von Driftzellen zu erzeugen.

Konstruktion dreidimensionaler Spuren

Analog zum Segmentierungsschritt werden in der zweiten Stufe vollständige Spuren aus den Segmenten gebildet. Da dies eine Extrapolation von einer Superlage zur nächsten erfordert und man auf dieser Größenskala nicht mehr von der Linearität der Bahnkurve also Näherung ausgehen kann, müssen eine Kreisbahn zu jedem Segment angepasst

werden. Um die Kreisbahnen zu schätzen, wird eine unverzerrte Näherung auf Basis der Methode der linearen kleinsten Quadrate verwendet, die in Spurfindungsanwendungen als Riemann-Fit bekannt ist. In dieser Arbeit wird die Riemann-Methode ausführlich untersucht und es wird gezeigt, dass sie insbesondere der konformen Abbildung überlegen ist. Darüber hinaus wird eine Verallgemeinerung aufgezeigt, die es möglich macht eine Anpassung an Driftkreise anstatt an eine Punktwolke vorzunehmen.

Der Algorithmus verläuft weiter wie folgt:

1. In der zweiten Stufe sind Tripel von Segmenten in der Kombination axial, stereo, axial, die optimale Wahl für die Knoten des Zellularautomaten, da ein Kreisbahn durch die beiden äußeren axialen Segmente ein verlässliche Rekonstruktion des in der Mitte befindlichen Stereosegments erlaubt.
2. Diese Segmenttripel besitzen daher eine dreidimensionale Position und eine dreidimensionale Bewegungsrichtung um eine Verbindung zum nächsten Tripel herzustellen oder zu verwerfen, wodurch die Kanten des Graphen bestimmt werden können.
3. Schließlich können die rekonstruierten Spuren als lange Pfad unter Verwendung des Zellularautomaten aus dem Graphen erzeugt werden.

Implementierung

Die durchgeführte Implementierung des dargestellten Algorithmus beinhaltet den erweiterten Riemann-Fit, den gewichten Zellulären Automaten und die diversen Zell– und Nachbarschaftskonstrukte. Des Weiteren strebt sie die Separation der kombinatorischen Aspekte von den frei einstellbaren Parametern an, sodass diese leicht austauschbar gegeneinander geprüft oder zu Optimierungszwecken mit Monte–Carlo–Informationen verglichen werden können. Letzteres ist unerlässlich, da die Parameter nicht a priori geschätzt werden können und es darüber hinaus schwer ist sie über die vielen Teilschritt hinweg ausschließlich anhand der finalen Spurfindungsqualität festzulegen. Die Monte–Carlo–Informationen wurden daher in der Entwicklung derart organisiert, dass die Evaluierung jedes Teilschrittes möglich ist. Vor Abschluss dieser Diplomarbeit konnten noch nicht alle Parameter auf ihre optimale Einstellung untersucht werden, was daher im weiteren Verlauf der Entwicklung getan werden muss.

Contents

Introduction

The honor of the new fascinating discoveries in high energy particle physics is most often attributed to high level physics analysts, but it is the builders of the detectors and accelerators as well as the developers of reconstruction software, who are ultimately responsible for the quality of the observations, that can be achieved. Each flaw in the lower layers of the data collection and reconstruction multiplies to the uncertainties in the final results and they have to be consequently diminished to obtain high accuracy results.

The human vision excels in the recognition of particle trajectories in graphic displays, but it has been a long time since the reconstruction of tracks from event photographies of particle decays by persons became unfeasible. Due to the huge amount of data generated in the billions of events by today's most complex measurement devices, called particle detectors, we need sophisticated and fast computer algorithms to carry out the pattern recognition for these experiments.

With the planned start in 2016 of the luminosity upgraded SuperKEKB asymmetric electron–positron–collider and the technologically upgraded Belle II experiment, the high energy research facility KEK in Japan strives to research yet undiscovered rare decay phenomena at a forty times higher luminosity. With an average number of nine particle tracks, electron–positron–collisions at the $\Upsilon(4S)$ resonance are known to present a clean event structures in the particle detectors, but the increased background from the intensified beams introduces new challenges to the tracking algorithms for the Belle II detector, which also seeks to exceed the tracking quality of the former Belle experiment.

In principle one can distinguish two different approaches to reconstruct particle trajectories from hits in a detector, which can be stated as follows:

Global tracking examines, if a template pattern is supported by the whole hit set in the physics event.

Local tracking is combining elementary hits bottom-up to tracks not necessarily obeying a predefined pattern.

In general global tracking is expected to be the faster approach, since it may obtain a reasonable result in a single step. An example of a global tracking technique is the widely known Hough transformation. In conjuncture with the divide and conquer paradigm in the Fast-Hough algorithm it makes the search for circular trajectories very performant, which is why it is often employed in the trigger stage of current particle

detectors. In contrast the local tracking is aggregating increasingly complex trajectory information stepwise from single hits in the detector to full track candidates, which makes it more susceptible to performance losses within each step, if the intermediate data is not handled efficiently. However in a local tracking algorithm the physical behavior of particles can be reflected in much more detail. Though being a major advantage it puts the burden of investigating a much more granulated space of decisions and parameters to the local tracking, making it laborious to create an optimal algorithm from first principles.

In the search for the most optimal tracking procedure for the central drift chamber of the Belle II experiment both techniques have been started to be analyzed at the Institute of Experimental Nuclear Physics in the Department of Physics of the Karlsruhe Institute of Technology.

This thesis seeks to prove the feasibility of a tracking algorithm fully based on local tracking techniques and to provide a solid base for further developments. As the concrete algorithm is being described in the Chapter 5, we state the major decision as well as some the possibilities that have been rejected. Pursuing the local tracking technique we will only rely on local relations between hits. Also we require as little dependence to the location of the interaction point as possible, hence we will not introduce a notion of inwards and outwards in the detector during the tracking, as well as no distinction between counterclockwise and clockwise. By limiting the range of the trajectory perceived at once the algorithm is less susceptible to multiple scattering effects and is also able to track particles curling inside the central drift chamber.

The division of the central drift chamber into superlayer of sense wires suggests the treatment of the local tracking in two stages. In the first segments limited by the superlayer boundaries will be combined from hits in the drift cells of the central drift chamber, while for the second stage these segments will be the basic building blocks of the full tracks to be produced.

Both stages are roughly treated equivalently. In order to limit the time consumed to carry out the necessary combinatorics on the elementary building blocks we refrain from using full backtracking of the event topology, but adopt the cellular automaton technique to determine long segments or tracks before building them. The cellular automaton usually uses discrete cell states, which we consider a too strong approximation, because all cells are treated equal. We will enhance the cellular automaton to a weighted form in order to reflect the substructure of the used cells and their local connections in more detail. The choice of the cell to use in the automaton is most important for the algorithm. We have sensibly chosen them to be triples of hits for the first stage, which we are referring to as facets, because of their triangular shape, and triples of segments for the second stage.

To judge the quality of the local connection we utilize fast fitting methods in each of the two stages. In the first we are relying on drift tangents to the drift circles exhibited by the drift time measurements in the central drift chamber as linear trajectories, while

we are implementing a fast circle fit, known to the tracking community as the Riemann fit, for the second stage. We will generalize the latter to fit to the drift circles instead of points and prove that it supersedes the often used conformal mapping method in Chapter 3. In order to make efficient calculations such as extrapolations with these two trajectory estimates as well as to justify the used fit models we also have to describe their mathematical representation in Chapter 2 in some detail.

The tracking algorithm shall figure out the subsets of hits that belong to the same particle track. The final fit of the track parameters will be carried out by the Kalman filter or the Dynamic Annealing Filter implementation in the *Genfit*[1] package.

Chapter 1
The Central Drift Chamber

A monk asked Zhaozhou, "Does a dog have Buddha nature or not?" Zhaozhou said, "Wú".

Kōan from the Gateless Gate

The task of the tracking algorithm developed in the scope of this thesis is to contribute to invented to help the reconstruction of events in the central drift chamber of the Belle II detector. In this chapter we give some context to the device and summarize the main technical features and measurement principles as far as we need them for the tracking.

1.1 Experimental context

1.1.1 SuperKEKB and the Belle II detector

The central drift chamber is a part of the future Belle II detector at the KEK high energy research facility in Japan. A scheme of the complete planned device is shown in Figure 1.1.

It serves as a detector at the SuperKEKB storage ring, which is in turn an upgrade of the asymmetric electron positron collider KEKB. As its predecessor it will operate at collision energies near the $\Upsilon(4S)$ resonance. The goal of these upgrades is to continue the investigation of B-physics, but at forty times higher luminosity, which may lead to the discovery of rare decay phenomena only accessible with a recorded event sample. However, with the higher luminosity the beam induced background will also increase and the Belle II detector has been carefully revised to account for it.

As all decay products are boosted by the beam momentum asymmetry, the detector has a distinct forward region where most of the particles fly. As a consequence the Belle II detector as well as the central drift chamber are slightly asymmetric to have a favorable acceptance of the produced particles.

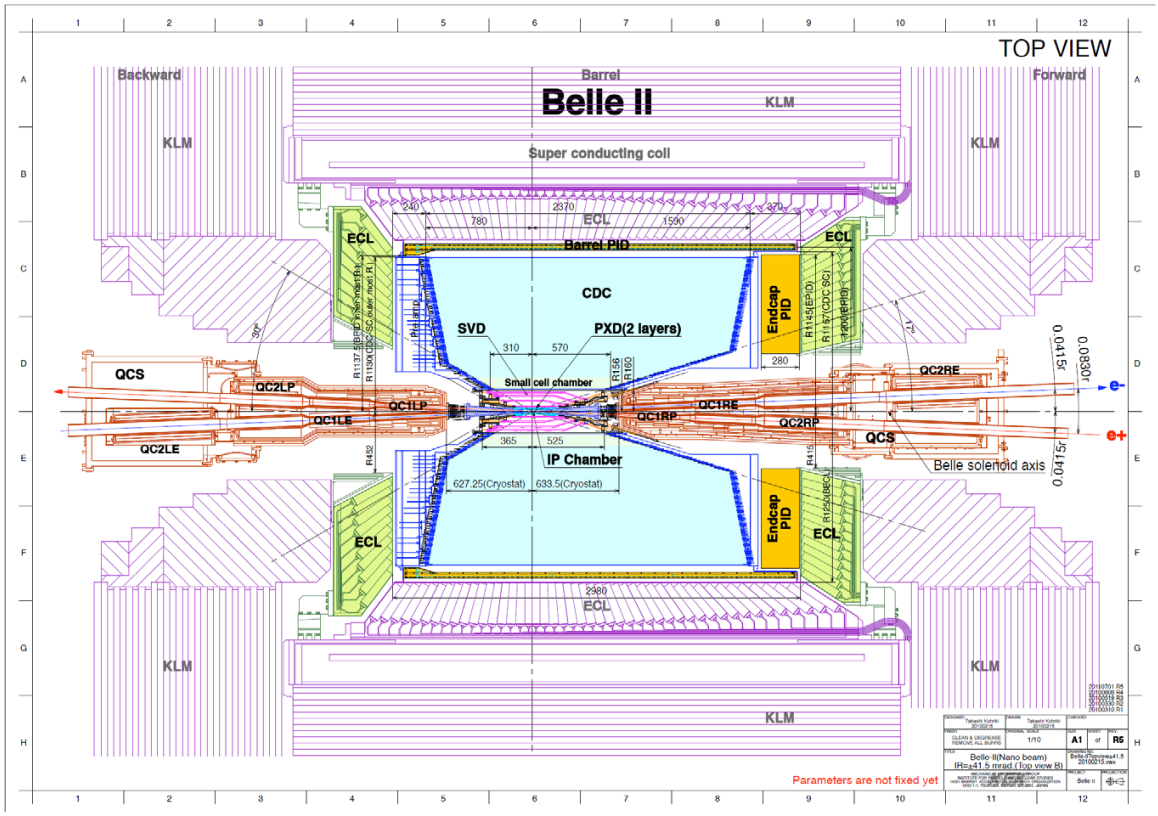


Figure 1.1: Top view of the planned Belle II detector. The central drift chamber (CDC) is marked blue.[2]

1.1.2 Purpose of the central drift chamber

The central drift chamber is designed to provide following information for each collision event

1. Momentum measurement of the particles

2. Trigger signals for the event recording

3. Particle identification for low momentum particles not reaching the outer detectors

This thesis is mostly concerned with the first aspect as we seek to find distinct particle trajectories in the event, which we can pass to a track fitting algorithm in order to determine their momentum.

Parameter	Value
Nominal magnetic field along beamline	$B = 1.5\,\mathrm{T}$
Gas mixture	50% helium - 50% ethane
Inner radius	160 mm
Minimal forward extend	570 mm
Minimal backward extend	310 mm
Outer radius	1130 mm
Maximal forward extend	1590 mm
Maximal backward extend	780 mm
Azimuthal acceptance	$17° - 150°$
Number of superlayers	9
Number of layers	56
Number of sense wires	14336

Table 1.1: Excerpt of the technical parameters of the central drift chamber [2].

1.2 Technical layout and detection principle

From a technical perspective the Belle II central drift chamber is a typical gas chamber detector with sense and field wires distributed in its volume as close as possible to the ionizing particle trajectories. It can be best understood as the direct successor to the drift chamber of the Belle detector, since it was operational without any problems over the whole data taking period [2, 3]. In the upgraded central drift chamber the same gas mixture, drift cell layout and operational voltage will be used. The main difference compared to the old detector is its larger dimension, the increased number of sense wires and their assembly into superlayers. Especially to counter the anticipated higher beam background near the beamline the inner most superlayer consists of smaller drift cells and contains two more layers than the other superlayers. Therefore it should perform as well as the other superlayers even if the occupancy is higher. A few parameters are summarized in Table 1.1.

1.2.1 Interaction with high energy particles

When a high energy charged particle passes through the drift chamber it ionizes the gas atoms. The energy deposit by the particle per travel length can be given by the famous Bethe-Bloch formula

$$-\left\langle \frac{dE}{dx} \right\rangle = Kz^2 \frac{Z}{A\beta^2} \left(\frac{1}{2} \frac{2m_e c^2 \beta^2 \gamma^2 E_{max}}{I^2} - \beta^2 - \frac{\delta(\beta\gamma)}{2} \right) \tag{1.1}$$

which lies at the heart of all high energy physics detection mechanisms. The energy is transferred to the electrons in the shell of the gas atoms and removes them from their

bound state. The generated free electrons can get accelerated by electric fields formed by the detector arrangement. The number of generated electrons is proportional to the deposited energy. In combination with the momentum of the particle this charge signal can be used to extract the mass of the particle and allows the identification of its type.

To resolve the momentum information of the particles the central drift chamber is permeated by a magnetic field of nominal field strength $B = 1.5\,\mathrm{T}$, which is aligned parallel to the beamline. It is generated by superconducting solenoid in the Belle II detector. Due to the interaction with the magnetic field particle trajectories through the central drift chamber are not straight but bent by the Lorentz force

$$\frac{\mathrm{d}\,\vec{p}}{\mathrm{d}\,\tau} = q \cdot \gamma \cdot \vec{\beta} \times \vec{B} \tag{1.2}$$

and resemble helices.

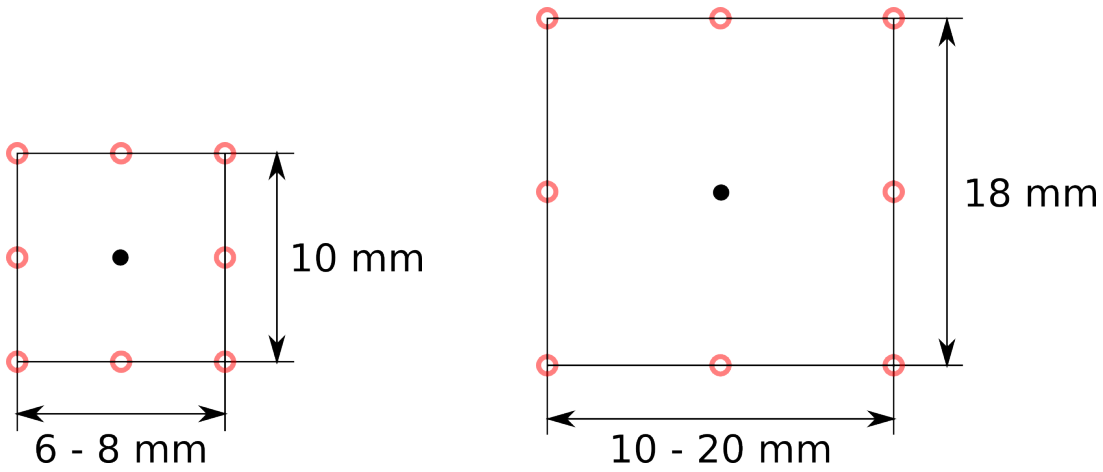
1.2.2 Drift cells

The electric field to collect the charge generated by the ionization is provided by the 14336 elementary drift cells in the central drift chamber. They are made of long wires, which are stretched between the forward and backward support plate with a rather high tension to minimize the gravitational sag. The wire arrangement as seen along the beamline is shown in Figure 1.2. A single sense wire made of gold plated tungsten is located in the middle to collect charges generated in the cell. They are connected to the readout electronics at the backward end plate at ground potential. At the border of the cell eight field wires are placed to shape the electric field in the cell and define a separating boundary to the next drift cells. The field wires are shared between neighboring cells. A high negative voltage U is connected to them to repel the electrons to the sense wires. The rather immobile gas ions are attracted by the field wires and get absorbed by them. The polarization of the field is chosen this way in order to let the more movable electrons generate a sharp current pulse in the sense wire. While the electrons approach the sense wire the field strength E rises according to

$$E \propto \frac{U}{r} \tag{1.3}$$

where r is the distance from the center of the wire. In the strong field next to the sense wires they get accelerated such that they can ionize more atoms on their own. In this way the drift cell serves as a preamplifier, which is inevitable for the generation of a detectable signal. In the following, we will use wire hits as synonym for drift cells that emitted a detected current pulse.

The magnetic field distorts the ideal straight approach of the electrons to the wire in addition to the imperfections near the boundaries of the drift cell. An instructive image generated from a simulation of this effect can be found in Figure 1.3.



(a) Small drift cell used in the inner most superlayer

(b) Large drift cell used for all but the inner most superlayer

Figure 1.2: Scheme of the drift cells in the central drift chamber as seen along the beamline. The left is the more compact version used in the inner most superlayer. The right more extend one is used for all other superlayers. Their width varies across a superlayer, since their growing perimeter has to be filled with the same amount of drift cells. The sense wire in the middle is marked with the filled black circle, while the eight field wires are marked red and non filled. Lines are not physical but for illustration only.

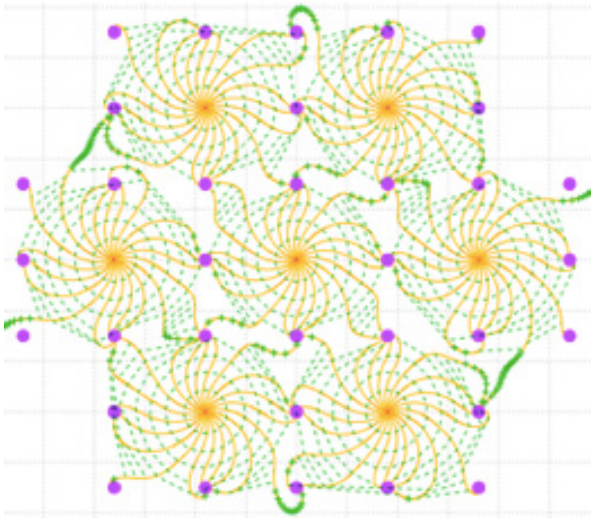


Figure 1.3: Drift path distortion from the magnetic field of seven neighboring drift cells. Violet filled circles represent the field wires, yellow curves are the drift paths of the electrons, green dashed lines are isochrones from the sense wires in the center of each drift cells[4].

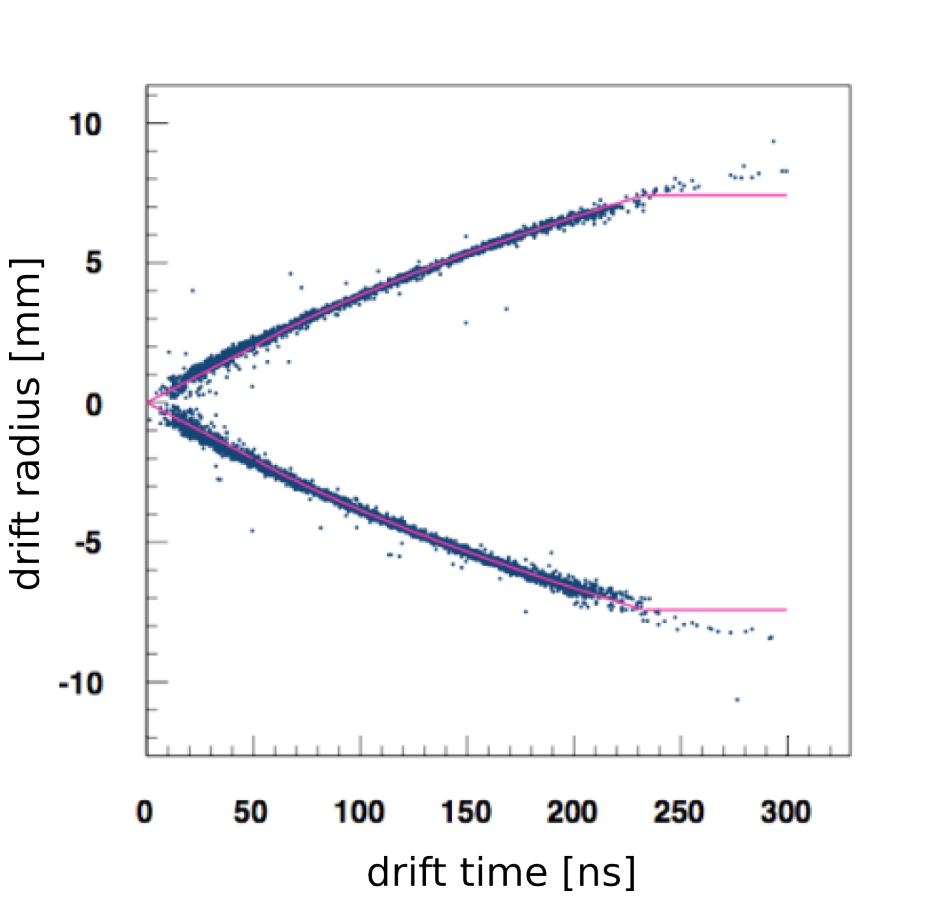


Figure 1.4: Drift relation for a small drift cell. The graph taken from the technical design report [2] illustrates a preliminary measurement.

1.2.3 Drift time measurement

The electron drift through the gas to the sense wires can be described as a diffusion process of the electrons with a characteristic drift velocity. Hence there will be a time delay between the ionization of the high energy particle we seek to detect and the electrons arriving at the sense wire. Knowing the drift velocity in the gas we can calculate distance of the high energy particle track from the wire. However we do not know the direction of the impending electrons on the sense wire, so the original interaction position is ambiguous but limited to a circle around the arrival position of the electrons on the sense wire. Still the drift time is taken from the closest approach of the trajectory to the wire, which also implies that the trajectory tangentially touches the drift circle. The radius of this drift circle is called the drift length. The relation between the drift time and the drift length is not exactly linear and has to be measured beforehand. Figure 1.4 illustrates a measurement of this relation prior to the Belle II design report. Nevertheless a roughly linear relation can be assumed with a drift

velocity of

$$v_{drift} = 4 \times 10^{-2} \, \frac{\text{mm}}{\text{ns}} \tag{1.4}$$

for the gas mixture in the central drift chamber. Hence the drift time can be as high as 500 ns for the largest drift cells.

The common start for all time measurements is the moment of the bunch crossing in the storage ring. Although it is non-trivial to synchronously distribute the timing signal from the bunch crossing frequency to all measurement devices throughout the detector, it is essentially a solvable problem. So we do not expect a noticeable time walk from a misunderstood primary interaction time within the 1 ns accuracy of the central drift chamber front end readout electronics [2].

However, we have to take two other delay times into account, which add to the measurement of the drift time. On the time scale of nanoseconds we cannot consider the particle arrival at the drift cell as instantaneous after it has been emitted from the interaction point, even if we considered it to fly at the speed of light. Indeed since the outer perimeter of the drift chamber is at 1130 mm particles need at least ≈ 4 ns to leave the detector. While we can attempt to correct for this delay, we have a similar delay for the forward or backward travel of the particle in the central drift chamber, which cannot be corrected before the tracking took place.

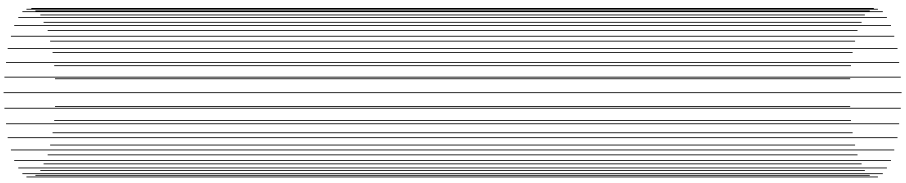
Also the propagation time of the current pulse in the wire is not entirely neglectable. The longest wires are approximately ≈ 2400 mm long, which can lead to an in–wire signal delay of maximal ≈ 8 ns.

In total there can be a delay time as large as ≈ 12 ns, which can account for a relative error of about 4%, if we assumed the average drift time to be half of the maximal possible drift time. The point of this argument was to give qualitative impression of the impact of the two mentioned delay times. Since we are going to make heavy use of the drift time during the tracking we have to be aware that all drift times of close by hits might suffer a common unknown delay due to the mentioned reasons.

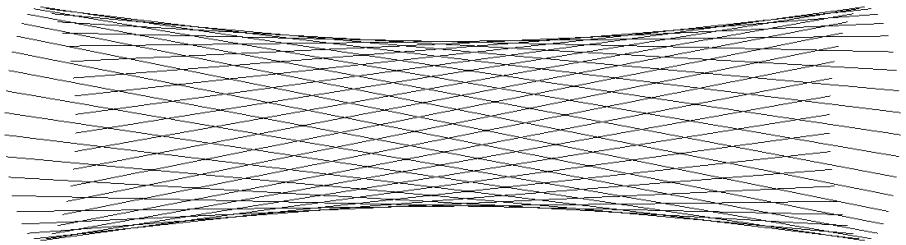
1.2.4 Layer structure

Drift cells sharing the same polar radius from the beamline are referred to as a layer. We distinguish two different types of layers depending on their alignment relative to the beamline. Sense wires in axial layers are all parallel to the beamline like illustrated in Figure 1.5a. A hit in this kind of wires immediately reveals the projected position of the particle along the beamline only limited by the drift circle ambiguity. No position information for the z coordinate can be derived from axial hits though.

Therefore the additional stereo layers have a wire arrangement skew to the beam axis in order to gather z information. The wires of stereo layers are lines taken from one-sheeted hyperboloid shapes like shown in Figure 1.5b. One can think of a stereo layer as generated from an axial layer by rotating one of the end plates relative to the other.



(a) An axial wire layer - sense wires are parallel to the beamline



(b) A stereo wire layer - sense wires are skewed to the beamline (exaggerated)

Figure 1.5: Geometry of the two types of wire layers in the central drift chamber.

The polar angle of stereo wires around the beam axis is not constant but changes from backward to forward end plate. Hence different stereo wires would be activated depending on position along the beamline, even if the particle was traveling on the same curve in the axial projection. In a two dimensional event display projected along the beamline, where the wires have to be represented by a reference position, the stereo layers show a distinct offset relative to the axial wire hits. In the detailed view in Figure 1.7, taken from a general event display in Figure 1.6, you can see the offset between axial and stereo layers for the actually smooth particle trajectories. The magnitude of the offset is essentially proportional to the z coordinate of the particle trajectory and can be used to reconstruct the third dimension. A mathematical description of the reference position taken for the display and of the mechanism for retrieving the z coordinate can be found in Chapter 2.

From the visualization we can also learn that axial and stereo layers are grouped together into superlayers. In total the central drift chamber has nine superlayers, which are alternating axial and stereo. Also there are two kinds of stereo layers called “U” and “V”, which differ in the direction of their twist. The superlayers parameters are summarized in Table 1.2. Note that the number of wire has been chosen to be a multiple of 32 to saturate the readout boards.

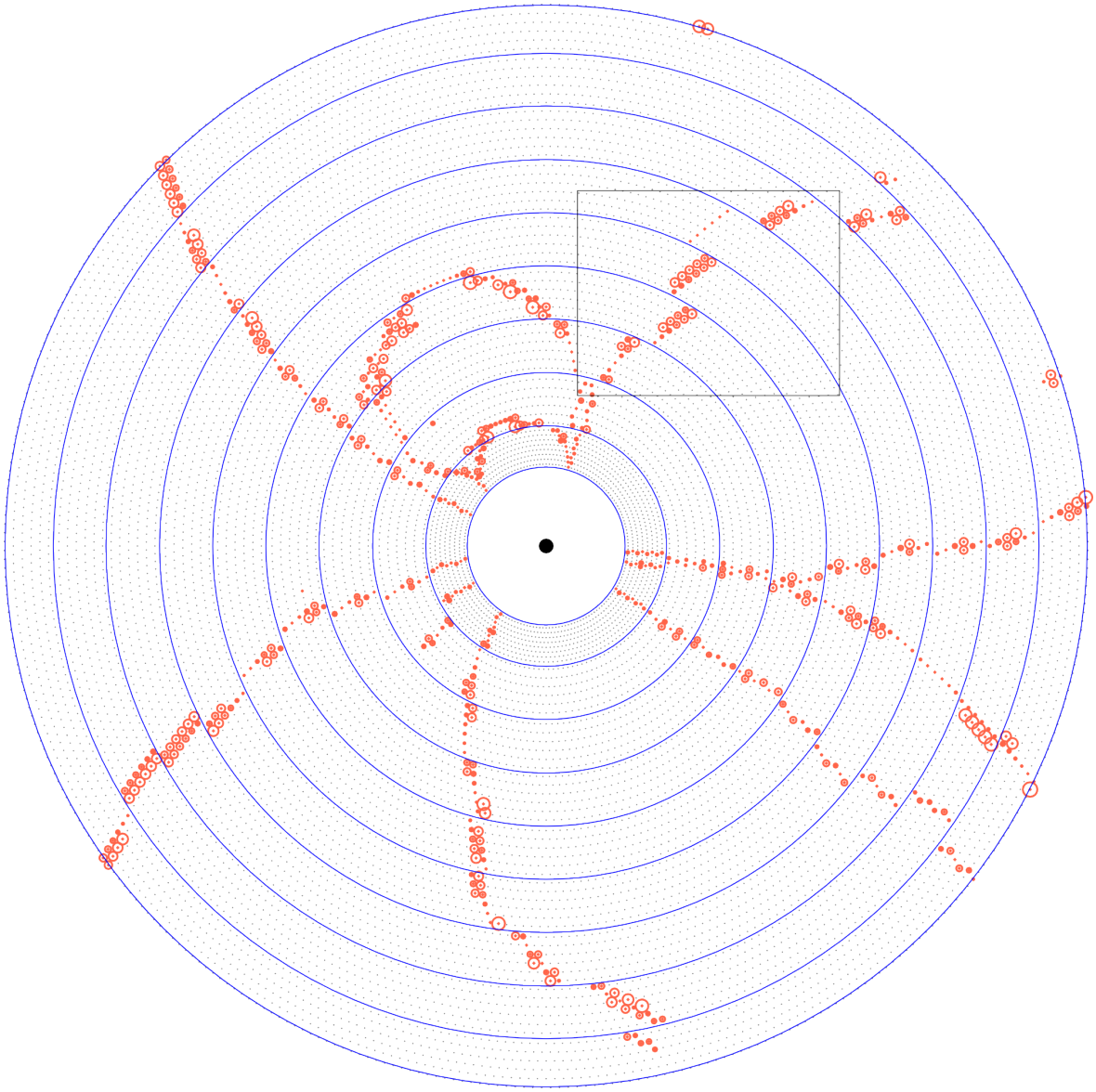


Figure 1.6: A typical event in the central drift chamber showing a decay of the $\Upsilon(4S)$ resonance without beam background. The decay products were generated with EvtGen[5] and their interaction with the detector was simulated by Geant4[6]. All sense wires are marked gray. The blue superlayer boundaries are illustrative and not physically present in the central drift chamber. Hit sense wires are drawn as red circles with radii representing their drift length. The black rectangular region is plotted again in Figure 1.7 to show the axial to stereo offset in more detail.

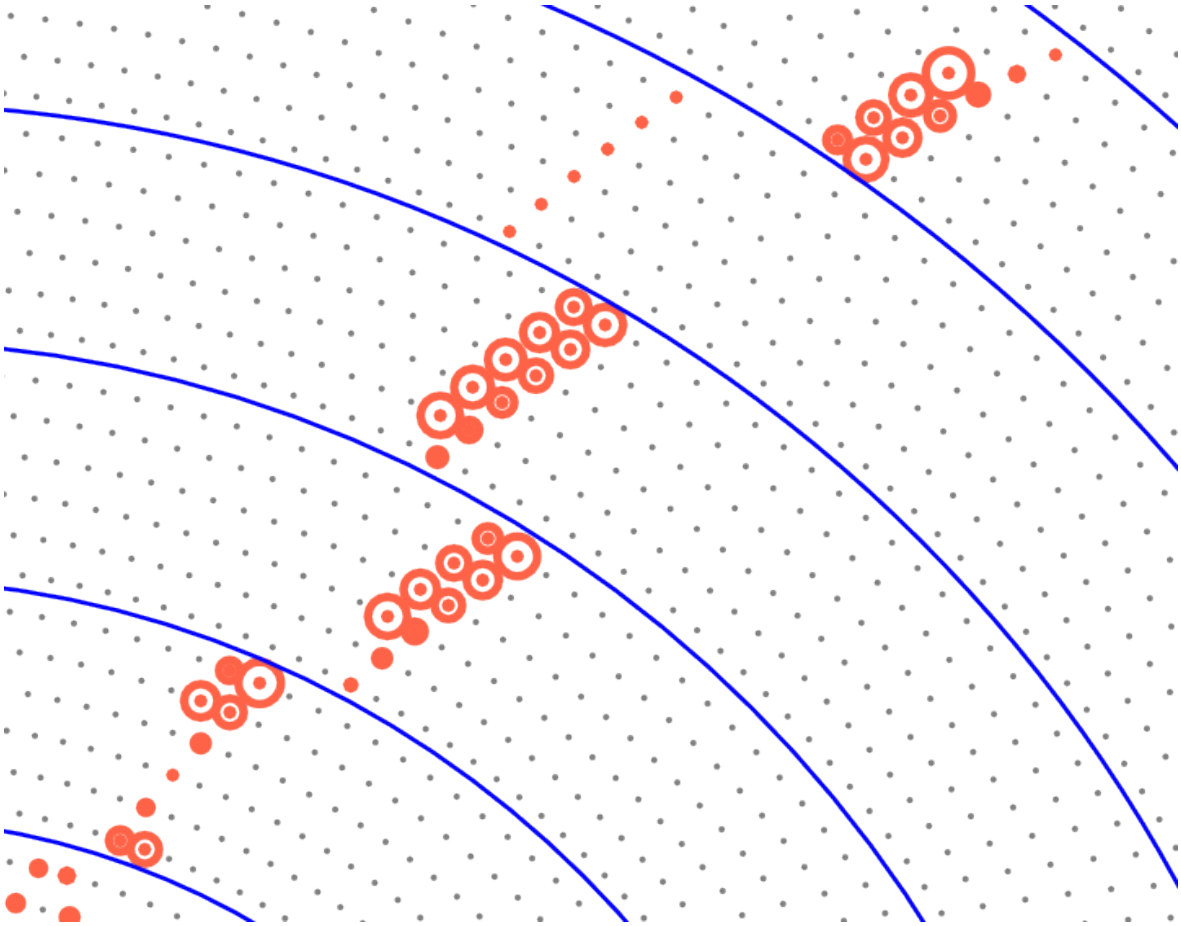


Figure 1.7: One particle track passing several superlayers. Track segments shown completely are contained in axial A, stereo V, axial A, stereo U, axial A superlayers in this order from inside outwards. The three axial parts are well aligned, while the stereo parts seem to be offset by a certain angle. This is the result of a particle trajectory, which is rather in the backward region of the central drift chamber, where the twist apart from the wire reference position shown is largest.

Superlayer number	Axial type	Number of layers	Number of wires per layer	Polar radius in mm	Stereo angle in mrad
1	Axial A	8	160	168.0 – 238.0	0 – 0
2	Stereo U	6	160	257.0 – 348.0	45.4 – 45.8
3	Axial A	6	192	365.2 – 455.7	0 – 0
4	Stereo V	6	224	476.9 – 566.9	-55.3 – -64.3
5	Axial A	6	256	584.1 – 674.1	0 – 0
6	Stereo U	6	288	695.3 – 785.3	63.1 – 70.0
7	Axial A	6	320	802.5 – 892.5	0 – 0
8	Stereo V	6	352	913.7 – 1003.7	-68.5 – -74.0
9	Axial A	6	384	1020.9 – 1111.4	0 – 0

Table 1.2: Overview of the superlayer arrangement and their parameter [2]

1.2.5 Momentum measurement

Momentum transverse to the beamline

Since axial wire hits present an undistorted image of the particle trajectories projected along the beam axis, the transverse component of the momentum can be determined from them. In this projection only a circle arc of the helix trajectory is visible. The radius of the circle r is related to the absolute transverse momentum and the magnetic field according to

$$|\vec{p}_t| = r \cdot B \cdot |q|, \tag{1.5}$$

where q is the charge of the particle. The direction of the momentum has to be taken tangential to the circle as it is different at each point. For high momentum particle we expect trajectory radii larger than the outer radius of the central drift chamber, but also low momentum particles curling several times inside of the central drift chamber are well possible.

Momentum parallel to the beamline

The motion parallel to the beamline is undisturbed by the magnetic field. Hence the travel distance along the beam axis increases linearly with time. As the travel time is not accurately measurable we have to relate the momentum parallel to the beam axis to the transverse momentum by a constant. This constant in turn is accessible by comparing the travel distance parallel to the beamline and the transverse travel distance. The transverse travel distance should be measured as the arc length on the circular trajectory, while the parallel travel distance has to be estimated through the

reconstruction using the stereo offset. If the two distance measures were linearly related like

$$z = m \cdot s + s_0 \tag{1.6}$$

we could obtain a parallel momentum of

$$p_z = m \cdot p_t \tag{1.7}$$

Note that the described method for retrieval of the momentum is a very simple approach to illustrate the feasibility of the measurements. For the final determination of the particle momenta we rely on the much more sophisticated Kalman filter algorithm of Genfit after the tracking stage. Nevertheless it is beneficial to have a interim momentum estimate for extrapolation purposes during tracking based on the method presented here.

Charge sign

The charge sign of the particle did not yet enter the described equations. The sign of charge directly relates to the clockwise or counterclockwise travel of the particle relative to the magnetic field. We conclude from the Lorentz force

$$\begin{aligned} q > 0 &\leftrightarrow \text{clockwise travel} \\ q < 0 &\leftrightarrow \text{counterclockwise travel} \end{aligned} \tag{1.8}$$

seen against the magnetic field vector. The decision, whether a particle is traveling counterclockwise or clockwise, is not easy to make from the projection along the wires. As long as we cannot observe a distinct direction of decreasing momentum, the static event record cannot tell forward from backward travel. In this case we have essentially to fall back to an argument by entropy and rely on the overall movement away from the interaction point to assume a travel direction.

1.2.6 Wire identification numbers

In order to refer to the individual sense wires, layers or superlayers there is a certain numbering scheme. We can distinguish five different numbers listed in Table 1.3.

Superlayer identification The superlayers can be indexed uniquely and continuously by their superlayer id. It increases outwards starting from zero.

Layer identification In the same sense the layers can be indexed by their continuous layer id, which also increase from the inside outwards. The complementary layer id is only continuous and unique within a specific superlayer. To identify a layer you can either give the superlayer id and the layer id or the continuous layer id.

Identification number (id)	Range		
Superlayer id	0 –	8	
Layer id	0 –	7	for superlayer id = 0
	0 –	5	for superlayer id = 1 − 8
Continuous layer id	0 –	55	
Wire id	0 –	159	for superlayer id = 0
	0 –	159	for superlayer id = 1
	0 –	191	for superlayer id = 2
	0 –	223	for superlayer id = 3
	0 –	255	for superlayer id = 4
	0 –	287	for superlayer id = 5
	0 –	319	for superlayer id = 6
	0 –	351	for superlayer id = 7
	0 –	383	for superlayer id = 8
Encoded wire id	0 –	35711	discontinuously

Table 1.3: Overview of the identification numbers attached to the wires, layers and superlayers

Sense wire identification For wires three possibilities for a full distinction are available. Within a layer wires can be denoted by their continuous and unique wire id. The wire ids increase counterclockwise seen against the magnetic field. All layers have their first wire with id zero close to the same transverse line, which is identical with the x axis of the detector. We can complete the full determination of a sense wire by appending the continuous layer id or the superlayer id and the layer id to the wire id. The later combination of three indices can also be blended into one index by

$$\text{encoded wire id} = 4096 \cdot \text{superlayer id} + 512 \cdot \text{layer id} + \text{wire id} \tag{1.9}$$

which saves storage space on disk and can be used as an unique discontinuous identification number over all sense wires.

1.2.7 Drift cell neighborhood

All layers of a superlayer have the same number of wires. Therefore drift cell can exhibit a near ordering which was chosen to be of hexagonal shape. Since this neighborhood is central to local tracking approaches, we introduce the following notation in order to refer to a specific neighbor. Considering Figure 1.8 we label the closest neighbors as listed in Table 1.4.

Naming scheme	Numbering scheme
Clockwise outwards	1
Clockwise	3
Clockwise inwards	5
Counterclockwise inwards	7
Counterclockwise	9
Counterclockwise outwards	11

Table 1.4: Naming scheme of the wire neighborhood.

The numbers are chosen to mark the positions of the neighbors on the normal twelve hour clock, where the twelve o'clock position points to the outside of the central drift chamber. Additionally we denote the twelve secondary neighbors of a drift cell by certain labels as illustrated in Figure 1.9. For brevity we use the numbers or call them by their o'clock position.

Note that the neighborhood cannot cross the superlayer boundaries, because the number of wires in the layer changes and the hexagonal shape cannot prevail. Hence if a sense wire is located near the superlayer boundary, not all neighbors can be present, which is most often the case for some of the secondary neighbors.

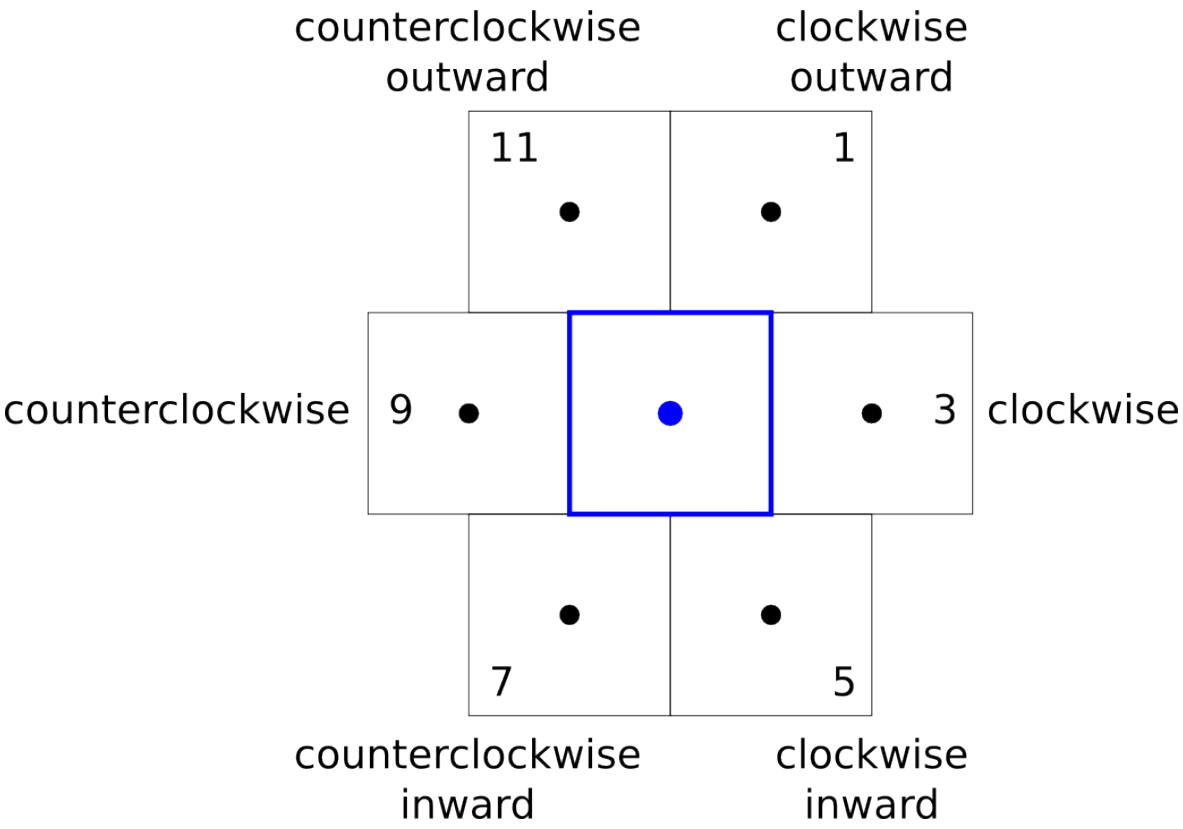


Figure 1.8: Closest neighborhood of the drift cell in the middle with labels given to the six closest neighbors. The interaction point is located to the bottom. The curvature of the layers has been neglected for the simplicity of the drawing.

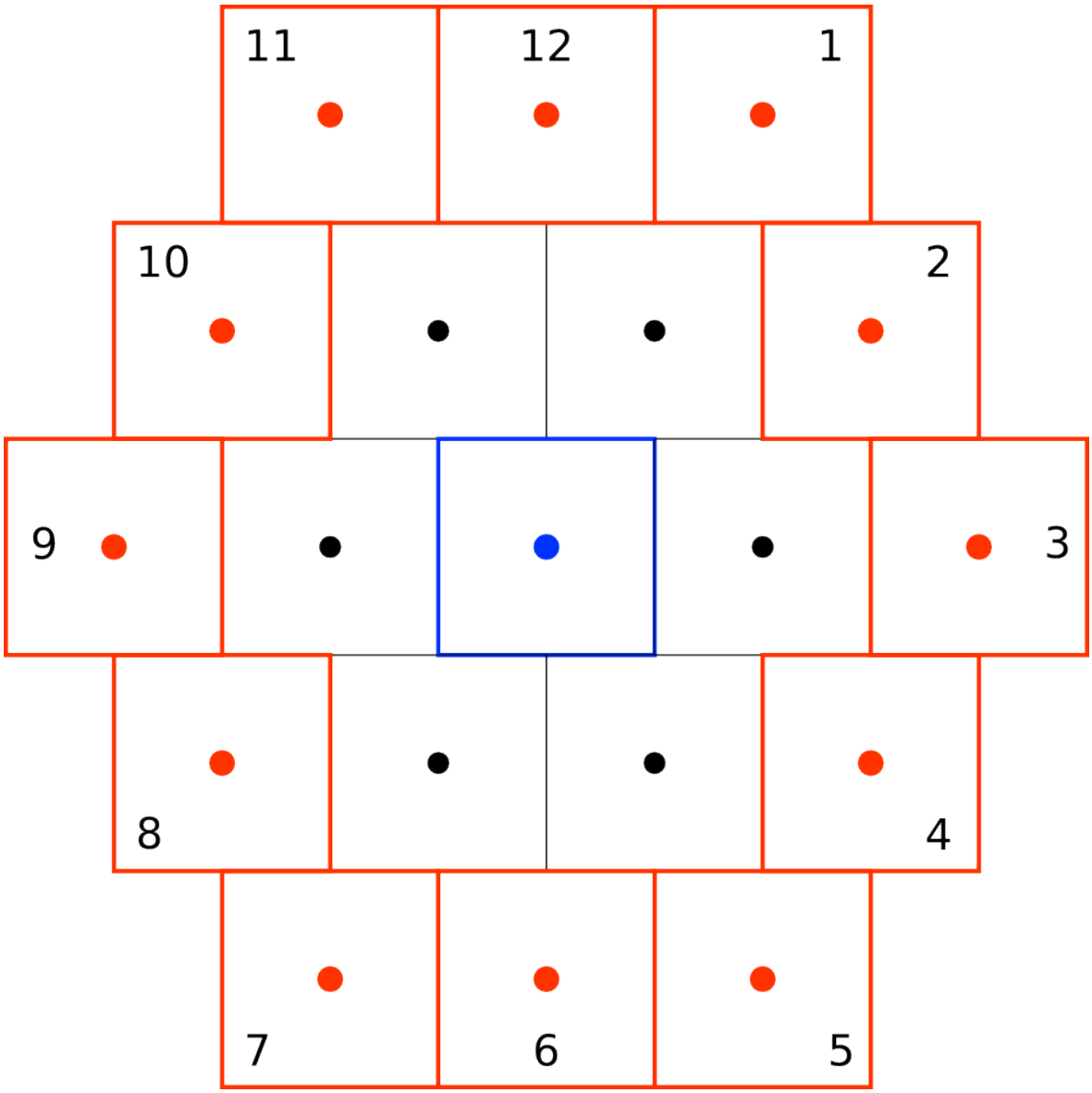


Figure 1.9: Secondary neighborhood in red of the drift cell in the middle with labels given to the twelve neighbors. The interaction point is located to the bottom. The curvature of the layers has been neglected for the simplicity of the drawing.

Chapter 2

Mathematical representations

> Every physicist who is any good knows six or seven different theoretical representations for exactly the same physics.
>
> Richard Feynman

Tracking is based on geometric description of the involved components, like particle trajectories, wire drift circles etc. In order to make useful and fast calculations, it is crucial to investigate the best representations regarding their simplicity, their generality, their parametrization and their usability in fits.

Most of the formulas given here are basic math. Since most of them have a one-to-one correspondence in the implementation code, it is still worth to not only state the final result, but give short derivations for some aspects. Also many definitions are stated explicitly to avoid later confusion and are not implied to be commonly known.

2.1 Coordinate systems

2.1.1 Laboratory coordinate system

The Belle II experiment has an agreed coordinate system [2]. The origin is at the design interaction point. Since we deal with an asymmetrical collider, the z coordinate is most naturally aligned with the beam of the higher energy. The forward region has positive z values, while the backward region has negative z values. The vertical axis away from the ground is identified with the y axis. The x axis is fixed to form a right hand coordinate system.

2.1.2 Transverse xy projection

Because the central drift chamber wires are essentially aligned in the z direction, the z coordinate of the hits is not known at the beginning of the tracking and we start tracking in the xy projection. We will often drop the z component of vectors, before we have not tried to reconstruct it. If we want to express this omission explicitly, we use $\vec{v}_{xy}$ or $\vec{v}_t$.

2.1.3 Switch of coordinate system

For local tracking it is beneficial to switch coordinates to the local situation not only to derive simpler solutions, but to keep the necessary computations to a minimum. Because of its special relation to the magnetic field and the wires, it is not necessary to change the z coordinate, but x and y coordinates are rather equivalent. We exploit this symmetry and rotate the coordinate system around the z axis as it is useful.

The matrix of an active counterclockwise rotation around the z axis is

$$\mathrm{Rot}(\gamma) = \begin{pmatrix} \cos\gamma & -\sin\gamma & 0 \\ \sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{pmatrix}. \tag{2.1}$$

If the transverse vector $\vec{c}$ is special in the situation at hand, we define a new orthonormal coordinate system with

$$\vec{c}_{\parallel} = \hat{c} := \frac{\vec{c}}{|\vec{c}|} \tag{2.2}$$

for the first axis. The second coordinate axis is implied to be the counterclockwise rotation of $\vec{c}_{\parallel}$.

$$\vec{c}_{\perp} = \mathrm{Rot}(\pi/2) \cdot \vec{c}_{\parallel} \tag{2.3}$$

The coordinates of a vector $\vec{x}$ in the new coordinate system are given by

$$x_{\parallel} = \vec{c}_{\parallel} \cdot \vec{x} \tag{2.4}$$

$$x_{\perp} = \vec{c}_{\perp} \cdot \vec{x}. \tag{2.5}$$

Another way to give the perpendicular coordinate is by introducing the two dimensional cross product[1]

$$\vec{c}_{\parallel} \times \vec{x} := \vec{c}_{\perp} \cdot \vec{x}. \tag{2.6}$$

[1]The two dimensional cross product is essentially the z coordinate of the three dimensional cross product.

For clarity we express these constructs in laboratory coordinates

$$\begin{aligned} \vec{c}_{\parallel} &= \begin{pmatrix} c_1 \\ c_2 \end{pmatrix} \qquad \vec{c}_{\parallel} \cdot \vec{x} = c_1 \cdot x_1 + c_2 \cdot x_2 \\ \vec{c}_{\perp} &= \begin{pmatrix} -c_2 \\ c_1 \end{pmatrix} \quad \vec{c}_{\parallel} \times \vec{x} = c_1 \cdot x_2 - c_2 \cdot x_1, \end{aligned} \tag{2.7}$$

although this is just to give some familiarity to the reader. In general we solely rely on the following trivial properties of the new coordinates

$$\begin{aligned} \vec{x} \cdot \vec{y} &= x_{\parallel} \cdot y_{\parallel} + x_{\perp} \cdot y_{\perp}, \\ \vec{x} \times \vec{y} &= x_{\parallel} \cdot y_{\perp} - x_{\parallel} \cdot y_{\perp}, \\ |\vec{x}|^2 &= x_{\parallel}^2 + x_{\perp}^2. \end{aligned} \tag{2.8}$$

If necessary we can translate back to laboratory coordinates with

$$\vec{x} = x_{\parallel} \cdot \vec{c}_{\parallel} + x_{\perp} \cdot \vec{c}_{\perp}. \tag{2.9}$$

2.2 Representation of lines

The most simple geometric object is the line. It has many desirable features and it is necessary to repeat some of them in order to generalize to a circle representation.

2.2.1 Normal representation

The normal representation of a general line in two dimensions [2] is

$$x \cdot n_x + y \cdot n_y + n_0 = 0 \quad \text{with} \quad n_x^2 + n_y^2 = 1, \tag{2.10}$$

where $\vec{n} = (n_x, n_y)$ is the unit normal vector to the line.

Distances

The given normalization is not required for the point set but gives a simple expression of the distance to any point $\vec{x}$ as

$$d_{abs}(\vec{x}) = \left| x \cdot n_x + y \cdot n_y + n_0 \right|. \tag{2.11}$$

By neglecting the absolute value

$$d(\vec{x}) = x \cdot n_x + y \cdot n_y + n_0 \tag{2.12}$$

[2] This omits the introduction of $y = m \cdot x + n$, since this is ill-parametrized for lines parallel to the y axis, hence losing generality and symmetry from the beginning.

we actually gain valuable information in the sign of the expression. If the distance is positive, the point $\vec{x}$ resides in the half plane the normal vector $\vec{n}$ is pointing to.

The key feature of signed distance from the line is its linearity in the parameters, which results in the known simplicity of line fits (see Chapter 3). Note also that the distance of the origin $\vec{o}$ to the line amounts to

$$d(\vec{o}) = n_0. \tag{2.13}$$

Hence it is only uncorrelated to one line parameter.

Orientation

The signed distance gives rise to an orientation and we define the *right* half plane relative to the line as all points having positive distance. The *left* half plane contains all points having negative distance.

$$\begin{aligned} d(\vec{x}) > 0 &\leftrightarrow \vec{x} \text{ is right of the line.} \\ d(\vec{x}) < 0 &\leftrightarrow \vec{x} \text{ is left of the line.} \end{aligned} \tag{2.14}$$

Direction of travel

The travel direction of a particle along the line seeing the right half plane on its right side must consistently be set to

$$\hat{p}_t = \begin{pmatrix} -n_y \\ n_x \end{pmatrix}. \tag{2.15}$$

We give a more formal derivation to ease a later argument for the circle (see Equation (2.63)). The distance $d(\vec{x})$ forms a scalar field in the plane. In this field the line is an equidistant curve of distance zero. From mathematics we know that the gradient of a scalar field is always perpendicular to its equipotential lines. The gradient of the distance field is

$$\nabla d(\vec{x}) = \begin{pmatrix} n_x \\ n_y \end{pmatrix}. \tag{2.16}$$

By rotating the gradient by $\pi/2$ we get a vector along the line. Whether to do that clockwise or counterclockwise depends on what is called right and left. To be consistent we choose the mathematical positive counterclockwise direction reproducing the result above

$$\hat{p}_t = \mathrm{Rot}(\pi/2) \cdot \nabla d(\vec{x}) = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} n_x \\ n_y \end{pmatrix} = \begin{pmatrix} -n_y \\ n_x \end{pmatrix}. \tag{2.17}$$

In general this gives only the direction of travel, but not a specific magnitude.

Right–left passage and distance to hits

Since the measurements in the central drift chamber are not points but drift circles we try to incorporate the drift radius a into the distance measure. In general the absolute distance has to be reduced according to

$$\delta_{abs}(\vec{x}) = \left| \left| x \cdot n_x + y \cdot n_y + n_0 \right| - a \right|, \tag{2.18}$$

which is not linear in the parameters. But we can also extend the signed distance with

$$\delta(\vec{x}) = x \cdot n_x + y \cdot n_y + n_0 \pm a, \tag{2.19}$$

retaining the linearity of the distance. But we introduced a decision, whether we want to add or subtract the drift radius to the signed distance, because this becomes undefined, when removing the inner absolute value. If the distance was originally negative, we have to add the drift radius to reduce the penalty for the hit. If the wire is on the left side of the line, we have to use plus in Equation (2.19), while we have to use minus for hits lying to the right. So additionally to the x and y coordinates of the wire and the drift length a, there is a sign variable α attributed to a hit, expressing the right or left passage of the particle relative to it. Since we want the correction to be a subtraction as in Equation (2.18), we define α such that

$$\delta(\vec{x}) = x \cdot n_x + y \cdot n_y + n_0 - \alpha \cdot a \tag{2.20}$$

is valid, hence

$$\alpha = \begin{cases} +1 & \text{hit in right half plane} \\ -1 & \text{hit in left half plane} \\ 0 & \text{unsure on which side} \end{cases}. \tag{2.21}$$

The last case $\alpha = 0$ erases the contribution of the drift length, so the distance is just reacting to the wire position as before. This should only occur, if the particle passes closely to the wire and we could easily confuse the passage side. In this case the drift length is small and its contribution insignificant. For further reference we state the mean of α at each wire to be

$$\langle \alpha \rangle = 0, \tag{2.22}$$

since we expect as many particles passing to the right of a wire as to the left.

The information on α is not measured by the detector, but has to be determined in some sort of pre-fit not considering the drift radii.

2.2.2 Parametric representation

The normal representation is only applicable in two dimensional space. To characterize a line in three dimensions we need to use the so called parameter representation according to

$$\vec{x}(\lambda) = \vec{m} + \lambda \cdot \vec{l}, \tag{2.23}$$

where λ is used in order to traverse all line positions. In addition to the normal representation we have an explicit support point $\vec{m}$ and a scale in the tangential vector $\vec{t}$ attached to the line. However a proper definition of right and left cannot be made in the three dimensional case.

Given two vectors $\vec{F}$ and $\vec{B}$ we get a line connecting both according to

$$\vec{x}(\lambda) = (1 - \lambda) \cdot \vec{B} + \lambda \cdot \vec{F} \tag{2.24}$$

with the following properties

$$\vec{l} = \vec{F} - \vec{B} \qquad \vec{m} = \vec{B} \qquad \vec{x}(0) = \vec{B} \qquad \vec{x}(1) = \vec{F}. \tag{2.25}$$

Additionally we give the lever arm form for the support point

$$\vec{m} = \frac{w_B \cdot \vec{F} + w_F \cdot \vec{B}}{w_B + w_F} \quad \text{where} \quad \lambda = \frac{w_B}{w_B + w_F}, \tag{2.26}$$

where the support vector becomes the center of mass, if the weights w_F and w_B were attached to the points $\vec{F}$ and $\vec{B}$.

2.2.3 Representation of the sense wires

For the matter of tracking we treat the wires in the central drift chamber as straight lines between the two joint points at the outer walls. Because of the limited wire tension and gravity the wires sag in the middle to a form known as the catenary. Still the linear approximation is a good starting point for the description.

Axial wires

Axial wires are parallel to the z-axis and we simply parametrize them using z as the independent variable

$$\vec{x}_A(z) = \begin{pmatrix} W_x \\ W_y \\ 0 \end{pmatrix} + z \cdot \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}. \tag{2.27}$$

Stereo wires

For stereo wires the situation is not as easy as for axial wires, because the x and y coordinate change with the z coordinate. Still we want to keep z as the independent variable. First we add a skew to the line according to

$$\vec{x}_S(z) = \begin{pmatrix} W_x \\ W_y \\ 0 \end{pmatrix} + z \cdot \begin{pmatrix} -\sigma \cdot W_y \\ \sigma \cdot W_x \\ 1 \end{pmatrix}, \tag{2.28}$$

where the *skew parameter* σ denotes the range of the rotation. The line is rotated around the axis, which is formed by the origin and the point of closest approach on the line to the origin. This keeps the distance of closest approach constant.

All wires in one layer share the same skew parameter σ, which leads to the hyperbolic shape of the wire layers shown in Figure 1.5b. It is rarely mentioned, that we have to take another parameter into account. The narrowest perimeter of the layer hyperboloid parametrized with Equation (2.28) always resides at $z = 0$. The design report [2] remains silent on this position and it should not assumed to be zero. It is rather up to the builders of the central drift chamber to fix it as it fits the construction. The preliminary geometry entered into the database of Belle II reveals the narrowest perimeter to be located at non-zero locations. We add a parameter W_z to Equation (2.28) to be able to move the wires parallel to the z direction as we wish.

$$\vec{x}(z) = \begin{pmatrix} W_x \\ W_y \\ W_z \end{pmatrix} + (z - W_z) \cdot \begin{pmatrix} -\sigma \cdot W_y \\ \sigma \cdot W_x \\ 1 \end{pmatrix} \tag{2.29}$$

Wire reference position

In this enhanced representation the reference point $\vec{W}$ marks the closest approach of the wire to the beam z-axis. In the xy projection this translates to

$$\vec{W}_{xy} \cdot \vec{l}_{xy} = 0, \tag{2.30}$$

meaning the reference point and the vector along the wire do not share a common component in this projection. Since this position is a special point on the sense wire in the context of its layer, $\vec{W}_{xy}$ will be the reference point used in the two dimensional tracking steps.

The skew parameter σ

The true benefit of the given representation using the skew parameter becomes visible by a slight cast to

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 1 & -\sigma \cdot \Delta z \\ \sigma \cdot \Delta z & 1 \end{pmatrix} \cdot \begin{pmatrix} W_x \\ W_y \end{pmatrix} \approx \mathrm{Rot}(\sigma \cdot \Delta z) \cdot \vec{W}_{xy}, \tag{2.31}$$

where we abbreviated

$$\Delta z = z - W_z. \tag{2.32}$$

The effect of a slight displacement in z direction away from W_z is a rotation of the wire reference position relative to the true position at $\vec{x}$.

Hence the skew σ is the rotation angle per z displacement. The unit of the skew can be given as

$$[\sigma] = \frac{\text{angle}}{\text{length}}. \tag{2.33}$$

Other resources, such as the Belle II design report [2], use the stereo angle ζ as the additional parameter instead and both are related according to

$$\tan\zeta = \sigma \cdot \left|\vec{W}_{xy}\right|. \tag{2.34}$$

However, the skew parameter is superior, because it has a clearer relation to the stereo offset in two dimensional event displays. Furthermore, it enters the representation above as a factor and does not have to obey to boundaries as the angle does.

From joint points to skew and reference position

In the database the sense wires are represented as the position of the two joint points at the front and back plate denoted by $\vec{F}$ and $\vec{B}$, from which we have to translate to the reference point / skew description. Without the full derivation we give the weights to be used in Equation (2.26)

$$w_F = \vec{F}_{xy} \cdot (\vec{F}_{xy} - \vec{B}_{xy}) \tag{2.35}$$

$$w_B = \vec{B}_{xy} \cdot (\vec{B}_{xy} - \vec{F}_{xy}), \tag{2.36}$$

which can easily be checked by substituting into Equation (2.30)

Give the reference point, the skew parameter is known to be

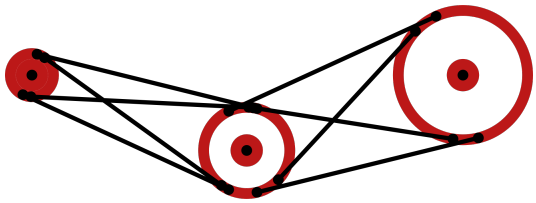
$$\sigma = \frac{\vec{W}_{xy} \times (\vec{F}_{xy} - \vec{B}_{xy})}{|\vec{W}_{xy}|^2 \cdot (F_z - B_z)}. \tag{2.37}$$

Wire end points

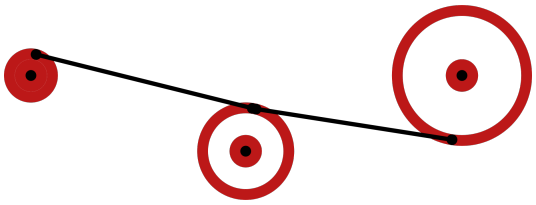
We also have to keep the end points of the wire, in case we want to check, whether a certain position, for instance on the trajectory, is already outside the central drift chamber. We only need to remember the z coordinates of the forward and backward joint point of the wire F_z and B_z, since

$$\vec{F} = \vec{x}_S(F_z) \quad \text{and} \quad \vec{B} = \vec{x}_S(B_z). \tag{2.38}$$

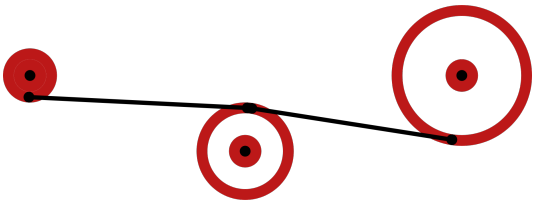
If we extrapolated a track to a certain layer in the xy projection, the only test left is for the z coordinate to be inside the central drift chamber. Hence, we do not need to keep the full vector.



(a) Three wire hits with four tangents connecting the pair to the left and four tangents connecting the pair to the right.



(b) One suitable combination of neighboring tangents making a reconstructed trajectory.



(c) Another suitable combination of neighboring tangents making a reconstructed trajectory.

Figure 2.1: Looking for possible tracks passing next to three neighboring wire hits we consider tangents to pairs of them. Accepting only a small angular deflection, we can obtain two possible reconstructed track segments in the situation illustrated

2.2.4 Tangents to drift circles

We argued in Chapter 1, that trajectories touch the drift circle of a wire hit tangentially. Hence we can use tangents to the drift circles to get linear approximations of the trajectories close to the hit.

A lot of a possible tangents can be constructed for a single hit, because it does not carry any direction of flight information on its own. In order to diminish these possibilities we construct tangents only in conjuncture with another nearby hit. These tangents shall to be interpreted as oriented possible linear trajectories of a particle causing these two hits, denoted by $_A$ and $_B$, in this specific order.

To justify to the notion of tangents to drift circles Figure 2.1 shows a possible application in tracking. The actual algorithm described in Chapter 5 uses the tangents in a different manner. Nevertheless, the figure correctly reflects an aspect of their actual use.

Passage information of tangents

From two general drift circles we can obtain four different tangents. Two of them pass on the same side of the drift circles and we refer to them as passing tangents. The other two have the hits placed on different sides of the line and we call them crossing tangents. Expressed in terms of the right–left passage variable α, passing tangents have either

$$\alpha_A = 1 \quad \alpha_B = 1 \quad \text{or} \quad \alpha_A = -1 \quad \alpha_B = -1, \tag{2.39}$$

while the crossing tangents have one of

$$\alpha_A = 1 \quad \alpha_B = -1 \quad \text{or} \quad \alpha_A = -1 \quad \alpha_B = 1, \tag{2.40}$$

Construction of tangents

Because we have to construct in each event, we explicitly express the most efficient way of computing them here. We are going to characterize a tangent by its two touch points to the drift circles. More concretely the touch points are described as displacements from the wire reference position with $\vec{t}_A$ and $\vec{t}_B$. The best coordinate system to express our solution is defined by the vector between the wire reference positions

$$\begin{aligned} \vec{c} &= \vec{W}_B - \vec{W}_A \\ \hat{c} &= \frac{\vec{W}_B - \vec{W}_A}{|\vec{W}_B - \vec{W}_A|}, \end{aligned} \tag{2.41}$$

because it provides a natural symmetry axis for the two different solutions in Equation (2.39) and Equation (2.40) respectively.

Crossing tangents We derive the components of the touching vectors $\vec{t}_{A/B}$ by the construction illustrated in Figure 2.2. We first note an important constant of proportionality

$$\kappa := \sin\gamma = \frac{a_A + a_B}{|\vec{c}|}, \tag{2.42}$$

which happens to be the sine of the marked angle in Figure 2.2. The cosine can be expressed by κ according to

$$\cos\gamma = \sqrt{1 - \kappa^2}. \tag{2.43}$$

The inverse trigonometric functions are computationally expensive and we refrain from calculating the angle and always use the ratio κ for speed.

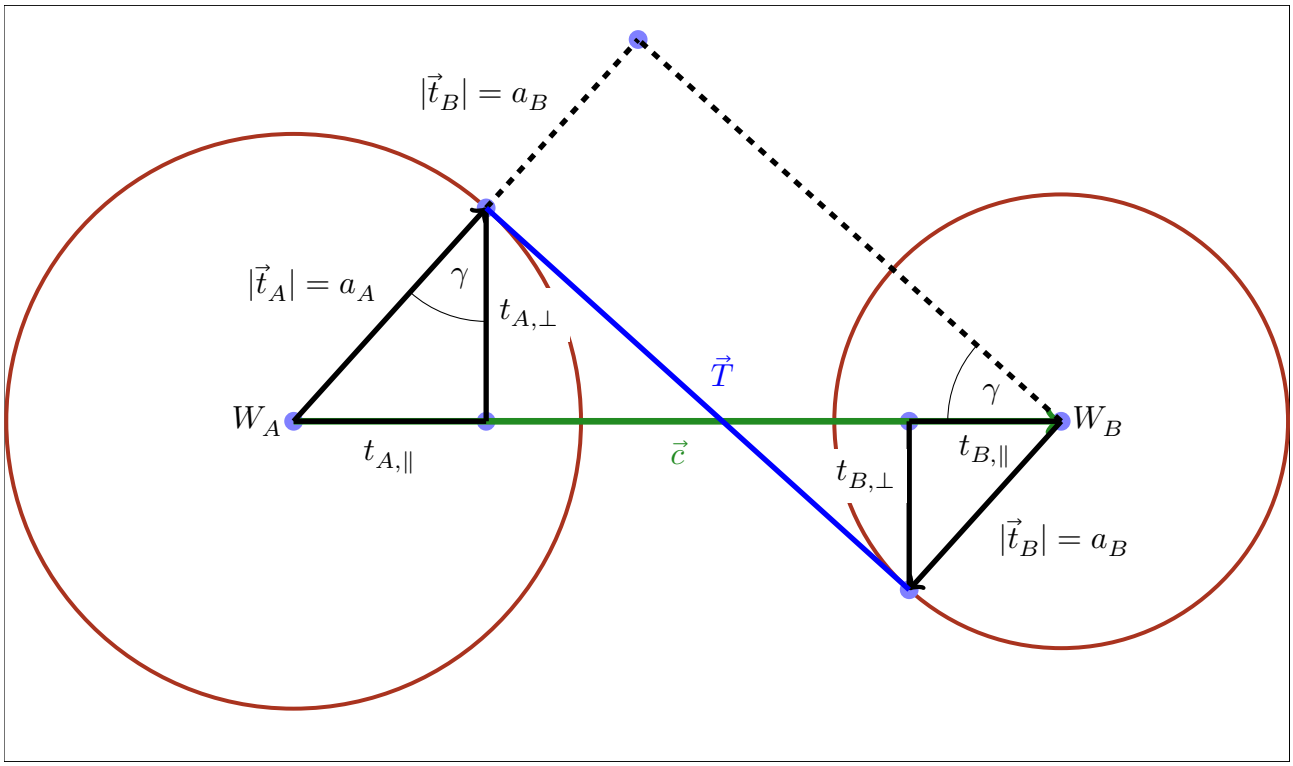


Figure 2.2: Construction of a crossing tangent. W_A and W_B are the wire reference positions and $\vec{c}$ is the vector connecting both. Vectors $\vec{t}_A$ and $\vec{t}_B$ are the displacements of the touch points from the wire reference position. Also their parallel and perpendicular component relative to coordinate system given by $\vec{c}$ are marked black. The dashed lines give the extension of $\vec{t}_A$ and a parallel to the sought tangent T through W_B. The two marked angles γ have equal magnitude.

With κ we can express the vector components making use of the theorem of intersecting lines.

$$\begin{aligned} t_{A,\parallel} &= \kappa \cdot a_A \\ t_{A,\perp} &= \pm\sqrt{1-\kappa^2} \cdot a_A \\ t_{B,\parallel} &= -\kappa \cdot a_B \\ t_{B,\perp} &= \mp\sqrt{1-\kappa^2} \cdot a_B \end{aligned} \tag{2.44}$$

These expressions give the two solutions for the crossing tangents as stated in Equation (2.40). We clearly see the symmetry between them, because they only differ by the change of sign in their perpendicular component. This corresponds to the reflection over the first axis of our carefully chosen coordinate system.

Passing tangents Similarly the passing tangents can be determined by considering Figure 2.3. Here κ as well as the two solutions are only slightly modified by some signs.

$$\kappa := \sin\gamma = \frac{a_A - a_B}{|\vec{c}|} \tag{2.45}$$

$$\begin{aligned} t_{A,\parallel} &= \kappa \cdot a_A \\ t_{A,\perp} &= \pm\sqrt{1-\kappa^2} \cdot a_A \\ t_{B,\parallel} &= \kappa \cdot a_B \\ t_{B,\perp} &= \pm\sqrt{1-\kappa^2} \cdot a_B \end{aligned} \tag{2.46}$$

The only difference is the replacement

$$a_B \to -a_B \tag{2.47}$$

to transform the solution from crossing to passing tangents.

Both tangents

The last observation can be exploited to merge the two solutions. The relevant signs can be encoded in the right–left passage variables $\alpha_{A/B}$ and we get to

$$\begin{aligned} \kappa &= \frac{\alpha_A \cdot a_A - \alpha_B \cdot a_B}{|\vec{c}|} \\ t_{i,\parallel} &= \kappa \cdot \alpha_i \cdot a_i \\ t_{i,\perp} &= \sqrt{1-\kappa^2} \cdot \alpha_i \cdot a_i \end{aligned} \tag{2.48}$$

In the end we have to consider some corner cases. The drift circle radii can be zero[3], which diminishes the number of valid tangents. A pair of hits with zero drift radii has

[3]The drift length a is often zero in the detector simulation to date.

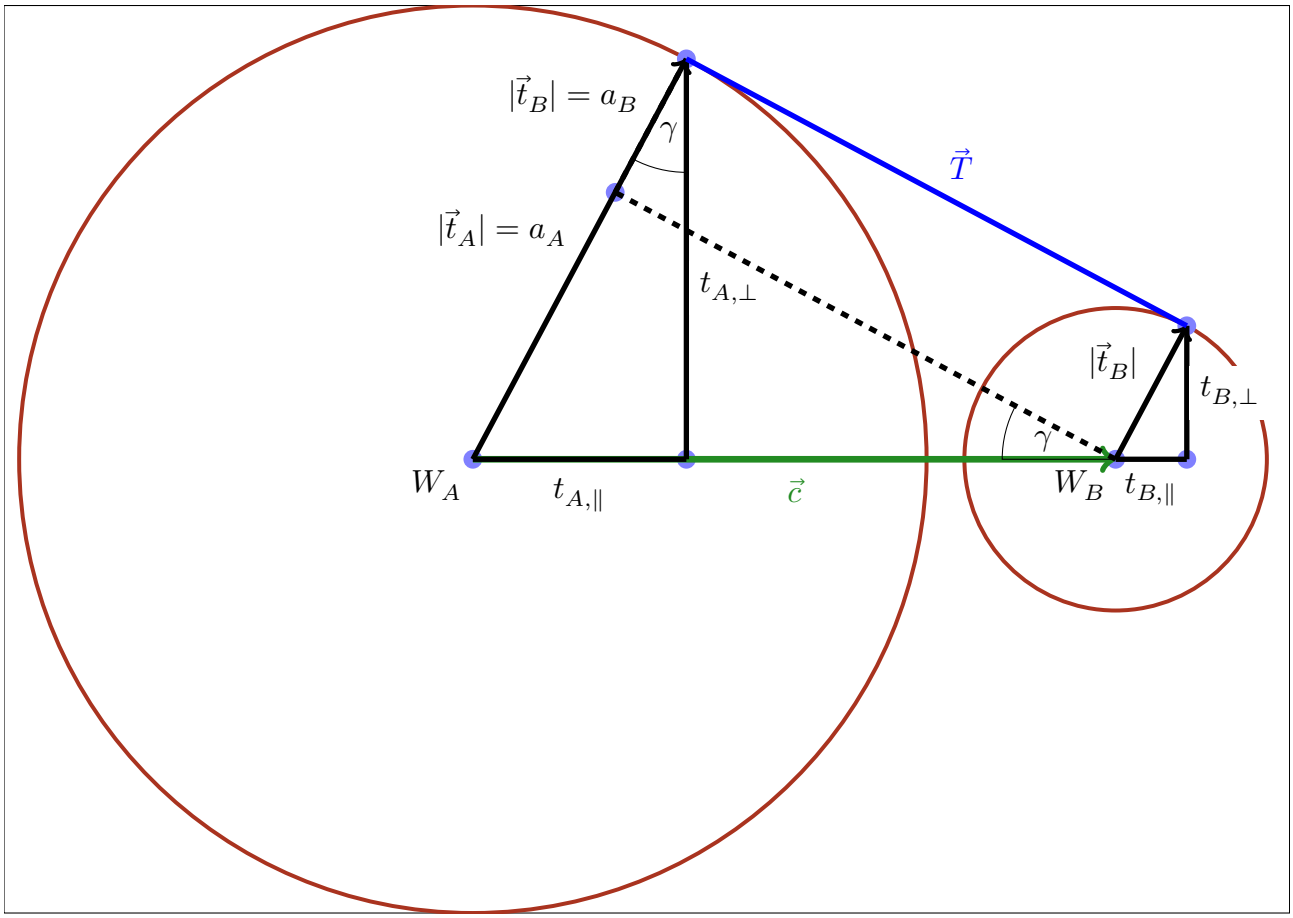


Figure 2.3: Construction of a passing tangent. The marked quantities are the same as in Figure 2.2. Less obvious is that, there is no dashed line for the extension of $\vec{t}_A$, since it is placed on the line segment inside the drift circle A. This leads directly to the additional minus sign in the solution.

only one tangent, which is the line between their wire positions. With one radius of the two set to zero we obtain only two tangents. The combined equations above handle both corner cases with ease, if we give $\alpha = 0$ for the corresponding drift radius instead of the two possibilities in Equation (2.39) and Equation (2.40).

Summary

The four major benefits of using tangents are that

1. tangents can be computed fast,
2. they are linear approximations of the track to be reconstructed,
3. contact points are candidates for likely positions of the primary ionization in the cell and
4. the right–left passage of hits can be resolved.

2.3 Representation of circles

Particles rarely present a linear trajectory, but circular ones with potentially low curvature. The goal is to find a smooth generalization of the normal line representation and its features to circles especially for using it as a fit model.

2.3.1 Natural representation

Stemming from the euclidean distance the most widely known representation of a circle is

$$(x - m_x)^2 + \left(y - m_y\right)^2 = r^2, \tag{2.49}$$

where the respective parameters $\vec{m} = (m_x, m_y)$ and r give the central point and the radius respectively.

Distances

This representation makes it easy to express the absolute distances of a point $\vec{x}$ to the circle according to

$$d_{abs}(\vec{v}) = \left| \sqrt{(v_x - m_x)^2 + \left(v_y - m_y\right)^2} - r \right|. \tag{2.50}$$

Again taking the absolute value hides some information

$$d(\vec{v}) = \sqrt{(p_x - m_x)^2 + \left(p_y - m_y\right)^2} - r. \tag{2.51}$$

A negative/positive value of d would denote that the point is inside/outside of the circle. This distance measure is nonlinear and not usable in a fast fit model.

Parametrization deficit

In high energy physics the observed particle momenta are rather high, which makes the trajectories more similar to a straight line than to a full circle. At this boundary the natural representation is ill–parametrized, because we have large parameters m_x and m_y, $m_x^2 + m_y^2$ to be precise, and a large radius r. At the same time the distance of the origin $\vec{o}$ to the circle

$$d(\vec{o}) = \sqrt{m_x^2 + m_y^2} - r \tag{2.52}$$

has to remain small. Hence we have revealed a correlation of the parameters, when trying to meet the origin with a low curvature circle as well as a numerical instability, because of the subtraction of two big numbers. The calculation of other distances from points to the circle suffers from the same instability.

To conclude we do not consider the natural representation to be a suitable description for higher energy particle tracks and we will not attempt to make use of it.

2.3.2 Generalized representation

During formulations of alternatives to the euclidean geometry, ultimately leading to the differential geometry and complex analysis providing the foundation of modern physics, mathematicians revealed the similarity of lines and circles combining both to the generalized circle representation

$$n_0 + x \cdot n_1 + y \cdot n_2 + \rho^2 \cdot n_3 = 0 \tag{2.53}$$

where $\rho^2 = x^2 + y^2$ is the polar radius of the given point. This representation visibly includes the line as a special case by setting $n_3 = 0$. Moreover for $n_0 = 0$ the origin lies on the circle.

Distances

In analogy to the line we could assume the distance measure related to this representation to be

$$\tilde{d} = n_0 + x \cdot n_1 + y \cdot n_2 + \rho^2 \cdot n_3 \tag{2.54}$$

The relation to the signed euclidean distance can easily be drawn.

$$\begin{aligned} d &= \sqrt{m_x^2 + m_y^2} - r \\ (d+r)^2 &= (x - m_x)^2 + (y - m_y)^2 \\ d^2 + 2 \cdot r \cdot d + r^2 &= x^2 + y^2 - 2m_x \cdot x - 2m_y \cdot y + m_x^2 + m_y^2 \\ d + \frac{d^2}{2r} &= \frac{1}{2r}\rho^2 - \frac{m_x}{r}x - \frac{m_y}{r}y + \frac{m_x^2 + m_y^2 - r^2}{2r} \end{aligned}$$

Identifying the new parameters n_i according to

$$n_0 = \frac{m_x^2 + m_y^2 - r^2}{2r} \qquad n_1 = -\frac{m_x}{r} \qquad n_2 = -\frac{m_y}{r} \qquad n_3 = \frac{1}{2r} \tag{2.55}$$

reveals the relation to the euclidean distance

$$d\left(1 + \frac{d}{2r}\right) = \tilde{d}. \tag{2.56}$$

The simplified distance measure $\tilde{d}$ is an approximation, which is accurate in the leading order of an expansion for $\frac{d}{r}$, to the euclidean distance. It is subjected to a relative error of

$$\sigma_{rel} = \frac{d}{2r}. \tag{2.57}$$

If we had the simplified distance $\tilde{d}$, we could calculate the euclidean distance from the quadratic equation

$$d^2 \cdot n_3 + d - \tilde{d} = 0. \tag{2.58}$$

The smaller root from

$$d_{1/2} = -\frac{1 \pm \sqrt{1 + 4\tilde{d} \cdot n_3}}{2n_3} \tag{2.59}$$

has to be taken as the correct solution, because to large root corresponds to the far point on the circle. The division by the small quantity n_3 is not numerically stable and we slightly recast to

$$d_2 = \frac{2\tilde{d}}{1 + \sqrt{1 + 4\tilde{d} \cdot n_3}} \tag{2.60}$$

to obtain a safely computable expression.

Normalization

The generalized circle representation acquired an additional parameter over the natural representation and thus the four parameters n_i are not independent, but present an invariant according to

$$n_1^2 + n_2^2 - 4n_0 \cdot n_3 = 1 \tag{2.61}$$

The considered Equation (2.53) yet lacks a normalization and we fix it using Equation (2.61) in order to avoid superfluous numerical constants. If we were interested in translating back to the natural representation we would use

$$r = \left| \frac{1}{2n_3} \right| \qquad m_x = -\frac{n_1}{2n_3} \qquad m_y = -\frac{n_2}{2n_3} \tag{2.62}$$

after we normalized correctly. Note that we anticipate a possible sign in the parameter n_3 and prevent it from propagating to the circle radius of the natural representation, which only accepts positive radii.

Direction of travel

Taking the derivation in Equation (2.17) as we did for the line the travel direction on the circle is

$$\vec{p}_t(\vec{x}) \propto \mathrm{Rot}(\pi/2) \cdot \nabla \tilde{d}(\vec{x}) = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} n_1 + 2n_3 \cdot x \\ n_2 + 2n_3 \cdot y \end{pmatrix} = \begin{pmatrix} -n_2 - 2n_3 \cdot y \\ n_1 + 2n_3 \cdot x \end{pmatrix} \tag{2.63}$$

p_t depends on the parameters n_i linearly, which means we can switch the travel direction by changing the sign of all four parameters n_i.

$$\begin{aligned} n_i &\to -n_i \\ p_t &\to -p_t \end{aligned} \tag{2.64}$$

Orientation

The observant reader might have noticed that the normalization still leaves a two fold ambiguity. By switching the sign of all parameters n_i the normalization does not change. This corresponds to two different orientations of the circle which we want to investigate here.

The distinction of which orientation is representing the clockwise or counterclockwise travel direction of the particle, can be made by using the gradient of the distance field.

$$\nabla \tilde{d}(\vec{x}) = \begin{pmatrix} n_1 + 2n_3 \cdot x \\ n_2 + 2n_3 \cdot y \end{pmatrix} = \vec{n} + 2n_3 \cdot \vec{x} \tag{2.65}$$

An important property of the gradient is its relation to the connecting vector $\vec{x} - \vec{m}$ from the circle center $\vec{m}$ to the point $\vec{x}$ at which we evaluate the gradient.

$$\begin{aligned} \vec{x} - \vec{m} &= \begin{pmatrix} x_1 + \frac{n_1}{2n_3} \\ x_2 + \frac{n_2}{2n_3} \end{pmatrix} \\ &= \frac{1}{2n_3} \begin{pmatrix} 2n_3 \cdot x_1 + n_1 \\ 2n_3 \cdot x_2 + n_2 \end{pmatrix} \\ &= \frac{1}{2n_3} \nabla \tilde{d}(\vec{x}) \end{aligned} \tag{2.66}$$

hence we have

$$\nabla \tilde{d}(\vec{x}) \parallel \vec{x} - \vec{m}. \tag{2.67}$$

Wherever one evaluates the gradient the resulting vector points to or away from the circle center.

$$\begin{aligned} &n_3 > 0 \leftrightarrow \nabla \tilde{d}(\vec{x}) \text{ points away from the circle center.} \\ &n_3 < 0 \leftrightarrow \nabla \tilde{d}(\vec{x}) \text{ points to the circle center.} \end{aligned} \tag{2.68}$$

The gradient picks up a minus sign during a switch of the orientation.

$$\nabla \tilde{d}(\vec{x}) \to -\nabla \tilde{d}(\vec{x}) \tag{2.69}$$

For counterclockwise travel the gradient should point away from the circle center, since a counterclockwise rotation of such a field results in the desired travel direction around the circle.

$$\begin{aligned} &n_3 > 0 \leftrightarrow \text{counterclockwise travel} \\ &n_3 < 0 \leftrightarrow \text{clockwise travel} \end{aligned} \tag{2.70}$$

The same definition of right and left as in the line case apply to the generalized circle. To the right of the trajectory all distances $\tilde{d}$ are positive, whereas they are negative on the left.

$$\begin{aligned} &\tilde{d}(\vec{x}) > 0 \leftrightarrow \vec{x} \text{ is right of the line.} \\ &\tilde{d}(\vec{x}) < 0 \leftrightarrow \vec{x} \text{ is left of the line.} \end{aligned} \tag{2.71}$$

Right left passage and distance to hits

Following the analogy to the line we want to correct the distance $\tilde{d}$ for the drift radius a. We assume the correction

$$\delta(\vec{x}) = d(\vec{x}) - \alpha \cdot a \tag{2.72}$$

with α having the same meaning as stated in Equation (2.21).

Omitting a lengthy derivation starting from Equation (2.51) we arrive at

$$\tilde{\delta} = \delta \cdot \left(1 + \frac{\alpha \cdot a}{r} + \frac{\delta}{2r}\right) = n_0 + x \cdot n_1 + y \cdot n_2 + (\rho^2 - \alpha^2 \cdot a^2) \cdot n_3 - \alpha \cdot a. \quad (2.73)$$

In addition to the anticipated $-\alpha \cdot a$ term the corrected distance measure $\tilde{\delta}$ acquires a small correction to the squared polar radius of the wire position. This correction might be neglected without harm, since the wire distance from the origin is much larger then the cell radius. The relative error of $\tilde{\delta}$ to the proper distance δ is

$$\sigma_{rel} = \frac{\alpha \cdot a}{r} + \frac{\delta}{2r}. \quad (2.74)$$

The two contributions to the relative error have different magnitudes as will be discussed in the following.

The proper distance δ of the hit to the trajectory can be assumed to be of the same order as the uncertainty in the drift length measurement. The later amounts to $< 10^{-1}$ cell radii. If we assume the trajectory of trackable particles to have a circle radius larger than 10 cell radii, the contribution of $\frac{\delta}{2r}$ to the relative error can be neglected, since it is of the order $\approx 10^{-2}$.

The contribution of $\frac{\alpha \cdot a}{r}$ is of order 10^{-1}, since a can be as large as a full cell radius. However it does not introduce a bias to the estimator $\tilde{\delta}$ of the true distance to the single drift circles, because of the vanishing mean of

$$\left\langle \delta \cdot \frac{\alpha \cdot a}{r} \right\rangle = 0 \quad (2.75)$$

since in addition to Equation (2.22), we expect δ to have a zero average and no correlation to the right–left passage. The latter is a desirable property for a fit model using the approximated distance (see Chapter 3).

2.3.3 Closest Approaches

Minimal distance approach

In order to make extrapolations to a hit we have to determine the point $\vec{x}$ on the circle which is closest to the wire position $\vec{v}$. The problem can be formulated as a minimization with a constraint

$$\min_{\vec{x}} |\vec{x} - \vec{v}|^2 \quad \text{subjected to} \quad \tilde{d} = n_0 + \vec{n} \cdot \vec{x} + n_3 |\vec{x}|^2 = 0. \quad (2.76)$$

Solving the constrained optimization with Lagrange's technique of multipliers makes the extended target function F

$$F(\vec{x}, \lambda) = |\vec{x} - \vec{v}|^2 + \lambda \cdot \tilde{d} \; = |\vec{x} - \vec{v}|^2 + \lambda \cdot \left(n_0 + \vec{n} \cdot \vec{x} + n_3 |\vec{x}|^2\right) \quad (2.77)$$

and we set the derivations of all three unknowns to zero

$$\frac{\partial F}{\partial \lambda} = \tilde{d}(\vec{x}) \stackrel{!}{=} 0 \tag{2.78}$$

$$\frac{\partial F}{\partial \vec{x}} = 2(\vec{x} - \vec{v}) + \lambda \cdot \nabla \tilde{d}(\vec{x}) \stackrel{!}{=} 0. \tag{2.79}$$

Again the gradient of the distance field has an important role. As we saw in Equation (2.67) it always points away from the center of the circle thus giving the direction along the line between circle center and point. The choice for the coordinate system will therefore best be made in terms of the gradient.

We choose our coordinate system to be along the vector

$$\vec{c} = \nabla \tilde{d}(\vec{v}) \qquad \vec{c}_{\parallel} = \frac{\nabla \tilde{d}(\vec{v})}{|\nabla \tilde{d}(\vec{v})|} \quad , \tag{2.80}$$

which reveals the perpendicular coordinate of $\vec{x}$ to be

$$x_{\perp} = \frac{\vec{n} \times \vec{v}}{|\nabla \tilde{d}(\vec{v})|} \tag{2.81}$$

after a few conversion steps. To get the parallel component we recast Equation (2.78) to the local coordinates

$$\begin{aligned} &\tilde{d}(\vec{x}) = n_0 + n_{\parallel} \cdot x_{\parallel} + n_{\perp} \cdot x_{\perp} + n_3 \cdot (x_{\parallel}^2 + x_{\perp}^2) \stackrel{!}{=} 0 \\ &\Rightarrow n_3 \cdot x_{\parallel}^2 + n_{\parallel} \cdot x_{\parallel} + (n_0 + n_{\perp} \cdot x_{\perp} + n_3 \cdot x_{\perp}^2) = 0. \end{aligned} \tag{2.82}$$

Solving this quadratic equation for $x_{\parallel}$ gives

$$x_{\parallel} = \begin{cases} \dfrac{1}{2n_3} \cdot \left(-n_{\parallel} \pm \sqrt{n_{\parallel}^2 - 4n_3 \cdot (n_0 + n_{\perp} \cdot x_{\perp} + n_3 \cdot x_{\perp}^2)} \right) & n_3 \neq 0 \\ -\dfrac{n_0 + n_{\perp} \cdot x_{\perp}}{n_{\parallel}} & n_3 = 0 \end{cases}, \tag{2.83}$$

where we also mention the line case $n_3 = 0$. Which of the two solutions is closer to the wire has to be decided by comparing the distances of the two candidates $\vec{x}_{1/2}$ to $\vec{v}$.

$$|\vec{x} - \vec{v}| = x_{\perp}^2 + x_{\parallel}^2 - 2x_{\parallel}v_{\parallel} - 2x_{\perp}v_{\perp} + |\vec{v}|^2 \tag{2.84}$$

This can be simplified by omitting terms shared between the two solutions to

$$\text{criterion} = x_{\parallel}^2 - 2x_{\parallel}v_{\parallel} = x_{\parallel} \cdot \left(x_{\parallel} - 2v_{\parallel} \right) \tag{2.85}$$

The closest approach is the candidate with the smaller criterion value.

Approach with same polar radius

In order to extrapolate to a certain wire layer we want to find a point on the circle which has the same polar radius as the layer. Because this problem has two solutions we still have to resolve the ambiguity by extrapolating to a specific wire $\vec{v}$ of the layer, which is expected to be close to the trajectory.

The constraint

$$|\vec{x}| = |\vec{v}| = \rho \tag{2.86}$$

in addition to Equation (2.76) simplifies the distance to minimize to

$$n_0 + n_3 \cdot \rho^2 + \vec{n} \cdot \vec{x} = 0. \tag{2.87}$$

The appropriate choice for the coordinate system is then obviously

$$\vec{c}_{\parallel} = \frac{\vec{n}}{|\vec{n}|} \tag{2.88}$$

and we achieve

$$x_{\parallel} = -\frac{n_0 + n_3 \cdot \rho^2}{|\vec{n}|} \tag{2.89}$$

The second coordinate has two solution given by the great $\Pi\upsilon\theta\alpha\gamma o\rho\alpha\varsigma$

$$x_{\perp,1/2} = \pm\sqrt{\rho^2 - x_{\parallel}^2} \tag{2.90}$$

Since $x_{\perp}^2$ is not different for the two candidate positions the selection criterion in Equation (2.85) reduces to

$$\text{criterion} = -2x_{\perp}v_{\perp} = \begin{cases} -2 \cdot \sqrt{\rho^2 - x_{\parallel}^2} \cdot v_{\perp} & \text{for } x_{\perp,1} \\ +2 \cdot \sqrt{\rho^2 - x_{\parallel}^2} \cdot v_{\perp} & \text{for } x_{\perp,2} \end{cases} \tag{2.91}$$

The candidate closer to the wire position $\vec{v}$ has the same sign in the perpendicular component $x_{\perp}$ as $v_{\perp}$.

2.3.4 Circle arc length

For fits in the sz-direction we need to measure the arc length s on the circle from one point $\vec{x}_1$ to a second point $\vec{x}_2$. The arc length is proportional to the opening angle ϕ on the circle according to

$$s = r \cdot \phi. \tag{2.92}$$

with the circle radius r as constant of proportionality.

To figure out the opening angle we use the gradient of the distance field to the circle, since it always points away from the circle center (see Equation (2.68)). The angle between the two gradient vectors at the points is the same as the opening angle of the arc. We consider

$$\tan \phi_{1\to 2} = \frac{\sin \phi_{1\to 2}}{\cos \phi_{1\to 2}} = \frac{\nabla \tilde{d}(\vec{x}_1) \times \nabla \tilde{d}(\vec{x}_2)}{\nabla \tilde{d}(\vec{x}_1) \cdot \nabla \tilde{d}(\vec{x}_2)} \tag{2.93}$$

and use the well–behaved arctangent[4].

$$\phi_{1\to 2} = \arctan \frac{\nabla \tilde{d}(\vec{x}_1) \times \nabla \tilde{d}(\vec{x}_2)}{\nabla \tilde{d}(\vec{x}_1) \cdot \nabla \tilde{d}(\vec{x}_2)} \tag{2.94}$$

to calculate the opening angle.

We purposely gave an orientation to the generalized circle. Switching the orientation of the circle should change the sign of the traveled arc length as well, since the flight direction is inverted. However the given formula of the opening angle does not change sign, since all sign changes in the argument of the arctangent cancel.

We have to reintroduce a sign by taking $\frac{1}{2n_3}$ instead of r as the constant in Equation (2.92) to obtain the correct arc length.

$$s_{1\to 2} = \frac{\phi_{1\to 2}}{2n_3} \tag{2.95}$$

For the special line case n_3 vanishes and the just derived travel distance is not applicable. Instead, we have to take the component parallel to the direction of travel according to

$$s_{1\to 2} = \hat{p}_t \cdot (\vec{x}_2 - \vec{x}_1) \tag{2.96}$$

or relating to the generalized circle parameters

$$s_{1\to 2} = \vec{n} \times (\vec{x}_2 - \vec{x}_1). \tag{2.97}$$

To summarize the travel distance on the generalized circle can be stated as

$$s_{1\to 2} = \begin{cases} \dfrac{\phi_{1\to 2}}{2n_3} & n_3 \neq 0 \\ \vec{n} \times (\vec{x}_2 - \vec{x}_1) & n_3 = 0 \end{cases}. \tag{2.98}$$

The formula is also applicable for points off the circle. We do not have to calculate the closest approach first to determine the travel distance.

[4] In program code this will be the $\mathtt{atan2}(\sin, \cos)$ function.

2.3.5 Reconstruction of the z coordinate

Only the position of the axial wires reflects the true xy position, while the reference positions of the stereo wires have to be rotated according to the z coordinate. If we find that a certain stereo hit belongs to a particle trajectory described as a generalized circle, we can use the stereo offset to reconstruct the z coordinate.

We essentially seek the position on the wire, which lies on the generalized circle in the xy projection and we have to calculate Δz in Equation (2.31) given Equation (2.53).

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 1 & -\sigma \cdot \Delta z \\ \sigma \cdot \Delta z & 1 \end{pmatrix} \cdot \begin{pmatrix} W_x \\ W_y \end{pmatrix} \quad \text{given} \quad n_0 + x \cdot n_1 + y \cdot n_2 + \rho^2 \cdot n_3 = 0 \tag{2.99}$$

Substituting the first into the second we get a quadratic equation for $\sigma\Delta z$.

$$\begin{aligned} n_0 + W_x \cdot n_1 + W_y \cdot n_2 + |W_{xy}|^2 \cdot n_3 + \\ \sigma \cdot \Delta z \cdot (W_x \cdot n_2 - W_y \cdot n_1) + \\ (\sigma \cdot \Delta z)^2 \cdot |W_{xy}|^2 \cdot n_3 = 0. \end{aligned} \tag{2.100}$$

Or using some already mentioned abbreviations

$$\tilde{d}(\vec{W}) + (\sigma \cdot \Delta z) \cdot (\vec{W}_{xy} \times \vec{n}) + (\sigma \cdot \Delta z)^2 \cdot |W_{xy}|^2 \cdot n_3 = 0 \tag{2.101}$$

The solution is

$$(\sigma \cdot \Delta z)_{1/2} = \begin{cases} \dfrac{-\vec{W}_{xy} \times \vec{n} \pm \sqrt{(\vec{W}_{xy} \times \vec{n})^2 - 4 \cdot |\vec{W}_{xy}|^2 \cdot n_3 \cdot \tilde{d}(\vec{W}_{xy})}}{2 \cdot |W_{xy}|^2 \cdot n_3} & n_3 \neq 0 \\ -\dfrac{\tilde{d}(\vec{W})}{\vec{W}_{xy} \times \vec{n}} & n_3 = 0 \end{cases} . \tag{2.102}$$

In the first case we use the smaller root assuming only small deviations from the reference position. We obtain the reconstructed xy position substituting $\sigma \cdot \Delta z$ into Equation (2.31) and the reconstructed z from Equation (2.32).

Chapter 3
Fast fitting

> Lots of things take time, and time was Momo's only form of wealth.
>
> Michael Ende

During the tracking procedure we need a quality indicator for how well the found track parts fit together, in order to reject track parts or to make extrapolations from one found part to a possible continuation. Since we are concerned with the track finding stage with a potentially large number of different track parts, the fits need to be fast with maybe some loss in accuracy.

Essentially there are two different fit problems, to which we need to find fast procedures.

1. Fitting circular trajectories with potentially low curvature to drift circles in the transverse plane.
2. Fitting the linear relation between the transverse and the z–travel distance as stated in Equation (1.6)

The later is obviously a line fit, which can be non-approximately solved by the fast linear least square method, since it only involves a single inversion of a small matrix. We only have to examine a feasible fast fit for circles.

The nonlinear euclidean distance from the circle could be used in nonlinear least square fit procedures, which are most exact, but too time consuming for tracking applications. We need to use an approximate method instead, which is preferably based on the faster linear least square method. Many approaches to this have been published [7–11], most often lacking simplicity and a clear description of the nature of approximation involved. During the investigations of the last chapter we derived an unbiased fit measure usable in a linear least square fit (compare Equation (2.54)). This measure has been used in tracking applications with great success [12]. We will recapitulate this method know as the Riemann fit to reflect the view of the primary authors and to give credit to them. With the additional background from the generalized circle representation we are however able to give a better justification of the approximation. Moreover, we will

extend the fit model from the original version, which fits to a set of points, to be able to fit to drift circles instead.

3.1 Method of least squares

3.1.1 General problem statement

If we want to model a dependent variable y by a function f with a number of free parameters n_j over a space of measurable variables $\vec{x}$ according to

$$\tilde{y} = f(\vec{x}|\vec{n}), \tag{3.1}$$

we need a method to determine the free parameters n_j from a number of observations $(y_i, \vec{x}_i)$. The least square method suggests estimating the parameters n_j by minimizing the sum of squared residuals: S

$$\min_{n_j} S(\vec{n}) \quad \text{where} \quad S(\vec{n}) = \sum_i (\tilde{y}_i - y_i)^2 \tag{3.2}$$

In this general formulation the solution rarely is expressible analytically and numerical minimization procedures have to be applied to iteratively minimize the sum of squared residuals, which is rather slow because of the large amount of calculations to be made.

3.1.2 Linear least square method

Simplifying the dependence f to a linear combination of the observation variables

$$\tilde{y} = \sum_j x_j \cdot n_j = \vec{x} \cdot \vec{n} \tag{3.3}$$

yields an analytically solvable problem. The sum S becomes

$$S(\vec{n}) = |\vec{y} - X \cdot \vec{n}|^2 \quad \text{with} \quad (X)_{ij} = x_{i,j}, \tag{3.4}$$

where X is the row matrix of the observed variables $\vec{x}_i$. Solving by partial derivation leads to the so called normal equation

$$\left(X^T X\right) \cdot \vec{n} = X^T \cdot \vec{y} \tag{3.5}$$

Its widely known formal solution [13]

$$\vec{n} = \left(X^T X\right)^{-1} X^T \cdot \vec{y} \tag{3.6}$$

is the least variance estimator for the parameters $\vec{n}$, if the observations $\vec{x}_i$ had uncorrelated, not necessarily normal distributed uncertainties of equal variance. This strong last claim is known as the Gauss-Markov theorem.

Numerical solutions

The numerical algorithms calculating the parameters $\vec{n}$ to minimize the target function S of the linear least square method almost never use the normal Equation (3.6), because the explicit matrix inversion poses numerical difficulties, especially if the matrix is ill conditioned. Feasible methods are the LR decomposition or QR decomposition [14]. For the tracking we are relying on another method based on the singular value decomposition (SVD) from the *Eigen* matrix library, which is proven to be exact and fast for matrices with only a few columns [15].

Scaling properties

A property of the linear least square method worth noting is the scale invariance relative to the input variables x_j. Replacing one of these variables with a scaled version according to

$$x_1 \to k \cdot x_1 \tag{3.7}$$

only alters the magnitude of the associated parameter

$$n_1 \to \frac{n_1}{k} \tag{3.8}$$

leaving the rest of the solution unchanged. This feature can be exploited to transform the input variables to the same range preventing some numerical difficulties.

Elimination of an intercept

Furthermore, if the model contained a constant input variable $x_0 = 1$, also called intercept or bias variable, we could eliminate it by using the deviations of the sample average

$$\begin{aligned} \Delta y &= y - \langle y \rangle \\ \Delta \vec{x} &= \vec{x} - \langle \vec{x} \rangle \end{aligned} \tag{3.9}$$

as input variables according to

$$\begin{aligned} y &= n_0 + \vec{x} \cdot \vec{n} \\ \langle \tilde{y} \rangle + \Delta \tilde{y} &= n_0 + (\langle \vec{x} \rangle + \Delta \vec{x}) \cdot \vec{n} \\ \Delta \tilde{y} &= \underbrace{n_0 + \langle \vec{x} \rangle \cdot \vec{n} - \langle y \rangle}_{n_0' = 0} + \Delta \vec{x} \cdot \vec{n} \end{aligned} \tag{3.10}$$

Taking the derivatives of Equation (3.4) reveals that the new intercept n_0' is zero, which reduces the optimization problem by one parameter.

To get the original n_0 back we use

$$n_0 = \langle y \rangle - \langle \vec{x} \rangle \cdot \vec{n} \tag{3.11}$$

Normalizations

A slight problem for the linear least square method arises if the dependent variable y vanishes for all observations. This happens for instance, if y is a distance desired to be zero. All solutions consistently yield zero for all free parameters n_j, which is the best solution to Equation (3.4).

$$0 = \tilde{y} = \vec{x} \cdot \vec{n} \rightarrow \vec{n} = 0 \tag{3.12}$$

Essentially this is due to the unknown scale of $\vec{y}$ in conjuncture with the scaling properties of the linear least square method. Still it is an undesired behavior if $\vec{n}$ has a normalization condition requiring at least one of the parameters n_j to be non-zero.

In case we know one of the parameters $n_0 \neq 0$ to be non-zero, we can overcome this limitation by a rearrangement of Equation (3.3) and fit rather to

$$x_0 = \sum_{j \neq 0} \left(-\frac{n_j}{n_0} \right) \cdot x_j \tag{3.13}$$

with new parameters $m_j = -\frac{n_j}{n_0}$. In this manner we only get the ratios of the original parameters. To get the proper value of the original parameters n_j the normalization has to be fixed manually.

If no specific parameter n_0 can be assumed to be non-zero, we have to consider Equation (3.4) again, which turns out to be

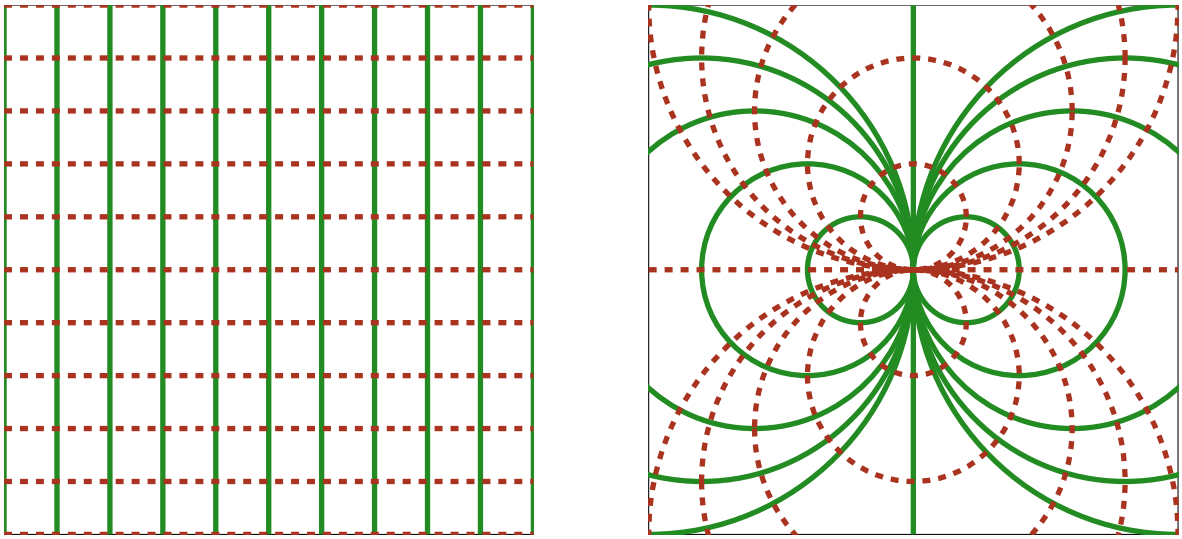
$$S = |X \cdot \vec{n}|^2 = \vec{n}^T \cdot X^T X \cdot \vec{n} \tag{3.14}$$

for $\vec{y} = 0$. Subjected to the specific normalization, the smallest S is obtained for the eigenvector $\vec{n}$ related to the smallest eigenvalue of $X^T X$.

3.2 Circle fits

Most fit problems, as the fit of a circle to points, are intrinsically non-linear. Still the linear least square method is useful, if we pose the right question. This means we have to find a mapping from the original nonlinear fit function to a form linear in the parameters, which introduces as little distortions as possible.

With the preparations done in Chapter 2 we can investigate which kinds of transformations are suitable in order to linearize the fit to a circle.



(a) Grid of lines in the original space

(b) Transformed grid in conformal space.

Figure 3.1: The conformal transformation maps lines (left) from the original space to circles through the origin (right) in conformal space and vice versa.

3.2.1 Conformal duality

A transformation widely used in tracking is the conformal map and we will shortly sketch its disadvantages here to consequently use a different transformation. The formula of the conformal map is given by

$$X = \frac{x}{x^2+y^2} = \frac{x}{\rho^2} \qquad Y = \frac{y}{x^2+y^2} = \frac{y}{\rho^2} \tag{3.15}$$

As a consequence the polar radius ρ of a point $\vec{x}$ is inverted according to

$$\mathrm{P} = \frac{1}{\rho} \tag{3.16}$$

Transformation of circles

Applied to a circle in the generalized representation (see Equation (2.53)) one can understand the effect of the transformation most easily.

$$\begin{array}{ccccccccc} N_0 & + & X\cdot N_1 & + & Y\cdot N_2 & + & \mathrm{P}^2\cdot N_3 & = & 0 \\ N_0 & + & \frac{x}{\rho^2}\cdot N_1 & + & \frac{y}{\rho^2}\cdot N_2 & + & \frac{x^2+y^2}{\rho^4}\cdot N_3 & = & 0 \\ \rho^2\cdot N_0 & + & x\cdot N_1 & + & y\cdot N_2 & + & N_3 & = & 0 \\ N_3 & + & x\cdot N_1 & + & y\cdot N_2 & + & \rho^2\cdot N_0 & = & 0 \end{array} \tag{3.17}$$

The set of all generalized circles is closed under the conformal transformation and the circle parameters are mapped to the conformal space according to

$$N_0 = n_3 \qquad N_1 = n_1 \qquad N_2 = n_2 \qquad N_3 = n_0 \tag{3.18}$$

by the exchange of n_0 and n_3. Hence a circle through the origin (with $n_0 = 0$) is transformed into a straight line (with $N_3 = 0$) in conformal space

$$x \cdot n_1 + y \cdot n_2 + \rho^2 \cdot n_3 = 0 \quad \rightarrow \quad N_0 + X \cdot N_1 + Y \cdot N_2 = 0 \tag{3.19}$$

An illustration of this property can be found in Figure 3.1.

Possible fit procedure

This property could be employed in a fitting procedure, if one was specifically interested in fitting circles through the origin by mapping them to the conformal space and fitting lines there. Since most particles come from the interaction point, which is the origin of the Belle II coordinate system, one could hope to cover the majority of trajectories. Many implementations for particle physics have been put forward in former high energy experiment [9] as well as for Belle II [16].

Distortions of the distance measure

However, the distance measure to the circle gets disastrously distorted according to

$$\begin{array}{rcccccccc} \tilde{D} & = & N_0 & + & X \cdot N_1 & + & Y \cdot N_2 & + & \mathrm{P}^2 \cdot N_3 \\ \tilde{D} & = & n_3 & + & \frac{x}{\rho^2} \cdot n_1 & + & \frac{y}{\rho^2} \cdot n_2 & + & \frac{x^2+y^2}{\rho^4} \cdot n_0 \\ \tilde{D} \cdot \rho^2 & = & \rho^2 \cdot n_3 & + & x \cdot n_1 & + & y \cdot n_2 & + & n_0 \\ \tilde{D} \cdot \rho^2 & = & n_0 & + & x \cdot n_1 & + & y \cdot n_2 & + & \rho^2 \cdot n_3 \end{array} \tag{3.20}$$

Hence the distance is getting scaled with $\frac{1}{\rho^2}$

$$\tilde{D} = \frac{\tilde{d}}{\rho^2} \tag{3.21}$$

While $\tilde{d}$ is the measure to be minimized, the line fit in conformal space is sensitive to $\tilde{D}$ instead. Therefore, we severely underestimate the importance of points far away from the origin in original space (high ρ value), while concentrating to much on the near origin points (low ρ value).

Despite the fact that one might want to degrade the importance of measurements far away from the origin to compensate for multiple scattering, a naive fit procedure of lines in the conformal space is error prone and needs to be corrected. An ansatz to

counter the distortion would be to weight each measurement according to its radius in original space with

$$w = \rho^4, \tag{3.22}$$

as it corrects the squared distance in conformal space to

$$w \cdot \tilde{D}^2 = \tilde{d}^2. \tag{3.23}$$

Inserting these weights to the linear least square method scales each measured point $\vec{X}$ in the conformal space by ρ^2, which is the inverse 3.19

$$N_0 + X \cdot N_1 + Y \cdot N_2 = 0 \quad \underset{\cdot\rho^2}{\rightarrow} \quad N_0 \cdot \rho^2 + x \cdot N_1 + y \cdot N_2 = 0, \tag{3.24}$$

negating the whole effect of the conformal map in the first place. The correctly weighted conformal mapping turns out to be equivalent to a special case of the Riemann fit, which we are about to present. We refrain from using the conformal method as a circle fit procedure.

3.2.2 Stereographic projection

To provide some further context to the Riemann fit we describe its direct predecessor in short. Another well known mapping from complex analysis is the stereographic projection. It can serve as a starting point for an unbiased fit to circles using the linear least square method [11].

The form of the stereographic projection with the least numerical constants can be given as

$$X = \frac{x}{\rho^2 + 1} \qquad Y = \frac{y}{\rho^2 + 1} \qquad Z = \frac{\rho^2}{\rho^2 + 1} \tag{3.25}$$

The projection maps the whole xy plane to a sphere in XYZ space with radius $\frac{1}{2}$ and center at $(0, 0, \frac{1}{2})$. The main property of the projection is to map generalized circles from the two dimensional space to circles on the sphere in three dimensional space (see Figure 3.2). Since intersections of the sphere and three dimensional planes can give all possible circles on the sphere, each generalized circle has a corresponding plane in XYZ space.

Possible fit procedure

The latter observation can be exploited for a fit procedure, where we fit planes in the three dimensional space with the linear least square method and translate back to the circle parameters afterwards.

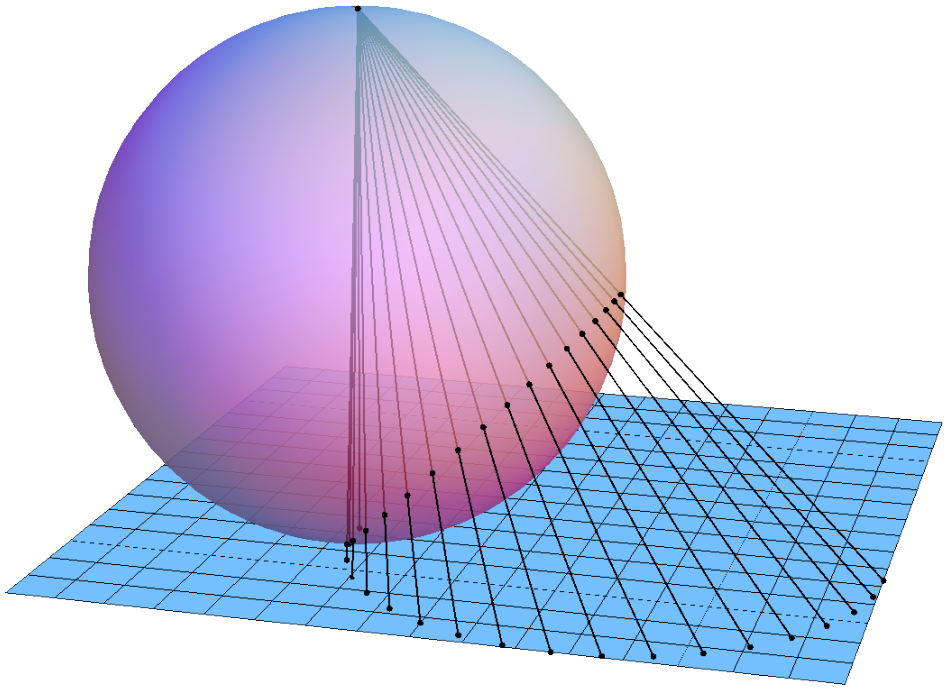


Figure 3.2: The stereographic projection can carried out by drawing projection lines from the north pol of the sphere placed over the two dimensional plane to be mapped. The mapped points lie, where the projection lines pierce through the sphere. All points from a two dimensional circle are projected into the same plane in three dimensional space.

In order to achieve an optimal estimator for the circle parameters, a careful investigation showed that weights depending on the polar radius according to

$$w = (\rho^2 + 1)^2 \tag{3.26}$$

have to be introduced for each measurement [17]. This is similar to the correction that has to be made to the conformal projection. Again the weights negate the purpose of the stereographic projection as they suggest to drop the common denominator in Equation (3.25).

3.2.3 Parabolic projection

By leaving out the common denominator from the stereographic projection we can circumvent the weighting corrections as has been demonstrated in [12].

$$X = x \qquad Y = y \qquad Z = \rho^2 \tag{3.27}$$

All points on the same generalized circle are still mapped to the same plane, however they do not form a circle in the three dimensional space anymore. The projection of generalized circles from two dimensional space can be all kinds of conic sections in three dimensional space (see Figure 3.3). A plane in the XYZ space consist of all points fulfilling

$$n_0 + X \cdot n_1 + Y \cdot n_2 + Z \cdot n_3 = 0 \tag{3.28}$$

Inserting the transformation immediately shows the relation of this projection to the generalized circle

$$n_0 + x \cdot n_1 + y \cdot n_2 + \rho^2 \cdot n_3 = 0 \tag{3.29}$$

which we extensively discussed in Chapter 2.

Riemann fit to points

Introducing the name *parabolic projection* or *Riemann fit* does little but to give a more visual meaning to the least square fit of the circle in the generalized circle representation, which we suggest here.

The only difference between the plane fit and the fit to the generalized circle is their different normalization. Planes are generally normalized according to

$$\vec{n} \cdot \vec{n} = 1, \tag{3.30}$$

while the generalized circles are normalized as already mentioned in Equation (2.61)

$$n_1^2 + n_2^2 - 4n_0 \cdot n_3 = 1, \tag{3.31}$$

which introduces a different scale factor invisible to the linear least square method.

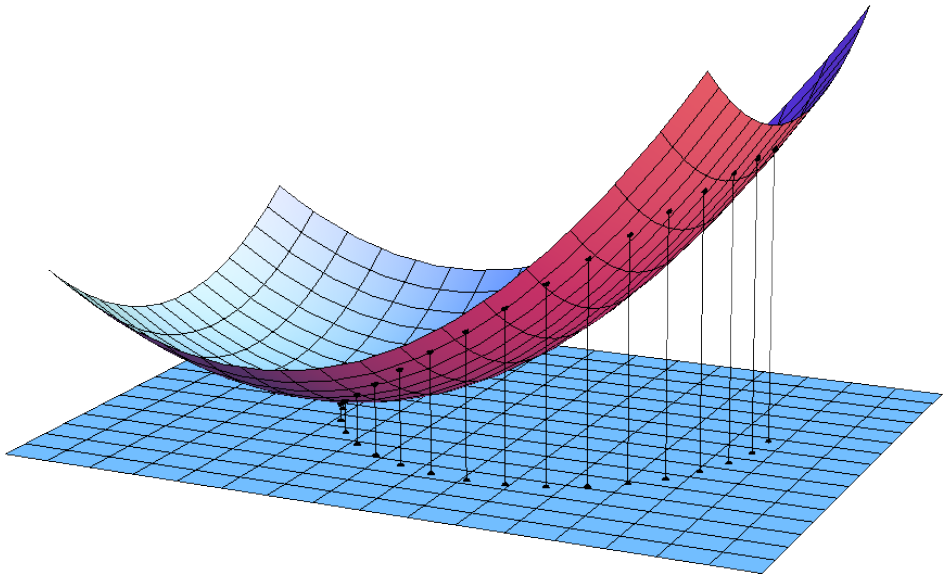


Figure 3.3: The parabolic projection can be understood as a lift of the points from the two dimensional space to a three dimensional paraboloid. Circles and lines in the two dimensional space are mapped to points lying in the same plane in three dimensional space.

So we state the linear relation f to be fitted by the linear least square method as

$$\tilde{d} = f(\vec{x}|\vec{n}) = n_0 + x_i \cdot n_1 + y_i \cdot n_2 + \rho_i^2 \cdot n_3 \tag{3.32}$$

Since the dependent variable $\tilde{d}$ is desired to be zero, we have to apply the modified version of the linear least square method as stated in Equation (3.14). Additionally, we have to fix the normalization and the orientation manually, since there is neither a fixed scale nor a fixed notion of right and left in the fit.

Moreover, it is simple to constrain the fit to circles through the origin by cutting the intercept variable according to

$$\tilde{d} = f(\vec{x}|\vec{n}) = x_i \cdot n_1 + y_i \cdot n_2 + \rho_i^2 \cdot n_3 \tag{3.33}$$

Riemann fit to drift circles

In Chapter 2 we showed that the distance $\tilde{d}$ can be corrected by the drift circle radii a given the right–left passage information α. If we already have a good assumption of the right–left passage information α for each hit, we choose $\tilde{\delta}$ from Equation (2.73) as the distance to be modeled by the linear least square fit

$$\tilde{\delta} = f(\vec{x}|\vec{n}) = n_0 + x_i \cdot n_1 + y_i \cdot n_2 + (\rho^2 - \alpha^2 \cdot a^2) \cdot n_3 - \alpha \cdot a \tag{3.34}$$

The last term does not contain a free parameter. As we argued with Equation (3.13), we can choose $\alpha \cdot a$ as the dependent variable to be modeled with the relation f instead of $\tilde{\delta}$ according to

$$\alpha \cdot a = f(\vec{x}|\vec{n}) = n_0 + x_i \cdot n_1 + y_i \cdot n_2 + (\rho^2 - \alpha^2 \cdot a^2) \cdot n_3 \tag{3.35}$$

The signed drift radii $\alpha \cdot a$ provide the scale to the linear least square method that was missing in the Riemann fit to points and we can apply the standard solution in Equation (3.6). Also the right left passage variable introduces an orientation to the fit. Consequently the circle parameters n_j represent the correct clockwise or counter-clockwise travel of the trajectory and no sign change to the normalization is needed.

Summary

We will use the Riemann fit in the tracking algorithm, because of the following reasons

1. The fit of generalized circles can be carried out by the fast linear least square method with four parameters.
2. No weighting needs to be done to correct the distances involved.
3. The approximated distance from trajectory circle to drift circle used in the fit is equivalent in leading order of $\frac{a}{r}$ (see Equation (2.73) or Equation (2.56)). Hence the fit is best for the low curvature circles we expect in tracking.
4. The approximated distance estimator is unbiased.
5. The magnitude of the error to the first order can be stated to be less than ten percent (see Equation (2.74)).
6. It can be enhanced to fit drift circles.
7. The circle acquires the correct orientation automatically in the enhanced fit.
8. It is constrainable to origin circles by neglecting the intercept variable. Also it is superior to the conformal map in this case.

To account for multiple scattering or drift time uncertainties the Riemann fit can be further improved by injecting a covariance matrix into the linear least square formula [12], if we consider it necessary.

Chapter 4

Cellular automation

> It always takes longer than you expect, even when you take into account Hofstadter's Law.
>
> Hofstadter's Law

Initial attempts to use combinatorial backtracking to identify long tracks have proven to be too time consuming. In order to avoid the combinatorial explosion, a commonly used technique in tracking applications is the use of an algorithm referred to as the cellular automaton [18, 19]. By applying it we can find long tracks in an event topology before actually building them. Hence there will be no need to backtrack all possible paths but only the promising ones, resulting in a tremendous speed up.

4.1 General problem statement

The cellular automaton as used in tracking is not to be confused with the formal definition from present informational science. The latter is an alternative to the Turing universal computing model, while for the tracking we are rather referring to the solution of the longest path problem, which just happens to keep intermediate values during the algorithm by assigning states to the cells used in it.

The longest path problem is normally stated in terms of graph theory. It amounts to finding the longest path in terms of vertices using only valid graph edges, while passing each vertex a single time at most. However in this general form the problem is known to be in the non-deterministic polynomial hard time complexity class *NP*. Since the solution of this problem takes time rising like $\mathcal{O}(n!)$, where n is roughly the number of vertices plus edges, it is not applicable in fast tracking in the general from. In tracking algorithms we fall back on the longest path problem on a *directed acyclic* graph, which is solvable in linear time $\mathcal{O}(n)$.

4.2 Unweighted algorithm

Since we want to introduce a generalization to the original cellular automaton algorithm used in tracking, we recapitulate the procedure in detail.

Starting from a given directed cycleless graph we want to figure out the longest path in it. The algorithm is carried out by assigning a state to each vertex (or cell, if we want to stick with the cellular automaton metaphor). The state should amount to the number of cells in the longest path starting from *this* cell including this cell.

We start with a random cell, which does not have a state assigned to it in order to figure out its correct state. From the currently examined cell we follow all edges and retrieve the states of the neighboring cells. We keep the highest retrieved state and store it augmented by one as the state of the examined cell. In case the neighboring cells do not have their cell state assigned yet, their state as to be determined recursively. Because we keep the already computed states stored in the cells, we are not backtracking whole parts of the graph multiple times, but visit each cell only one time in order to calculate its state. For this reason the time complexity is linearly bound by the number of cells and the number of edges. Also note that the acyclic requirement of the graph guarantees that the recursion is not infinite. The procedure is illustrated in Figure 4.1.

The final state S of cell i can be given as an equation:

$$S_i = 1 + \max_{\text{neighbor } j} S_j \tag{4.1}$$

Having assigned all states we can start from the highest of all states in the graph and pick up the neighbors, which have a state one less then the current cell. Hence we are following cells, that were responsible for the high state of the current cell.

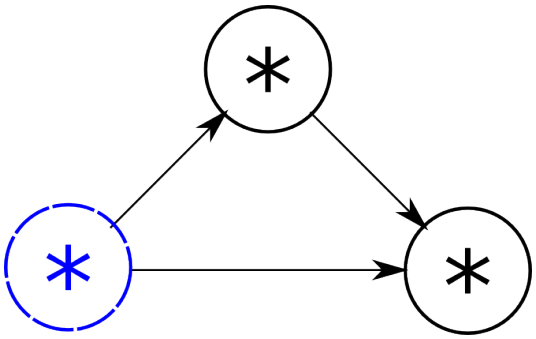
In tracking we can not be sure, if the given directed graph does contain any cycles or not. In order to detect a cycle and to prevent infinite loops we mark the cells we are currently traversing in the recursion cycle by an additional state variable. If a cycle occurs the method to generate sensible paths from it has to be altered.

The concrete connection from the cellular automaton to the tracking application is easily drawn.

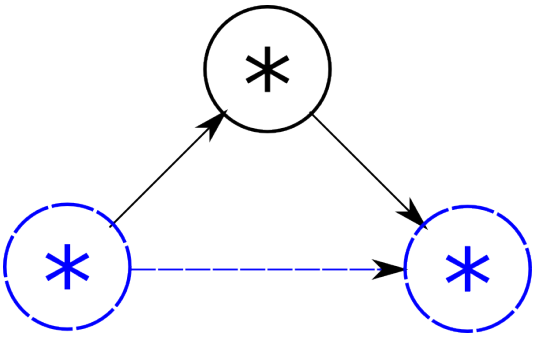
1. Paths in the graph are related to track parts or whole tracks.

2. Vertices are possible positions of the particle based on hit information.

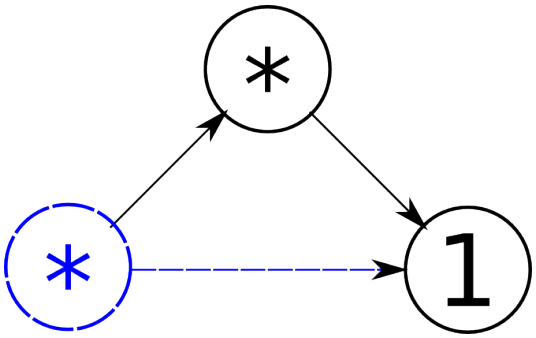
3. Edges are possible transitions of particles from one position to another again based on the compatibility of the hit information.



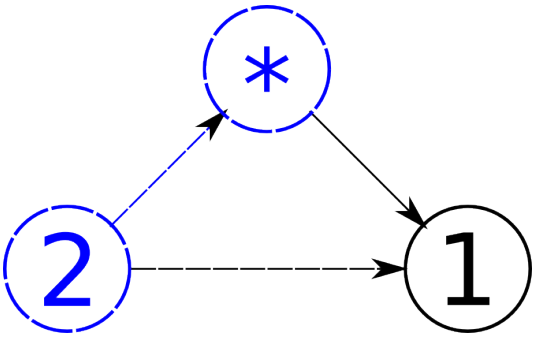
(a) Starting with a random unassigned cell (blue), …



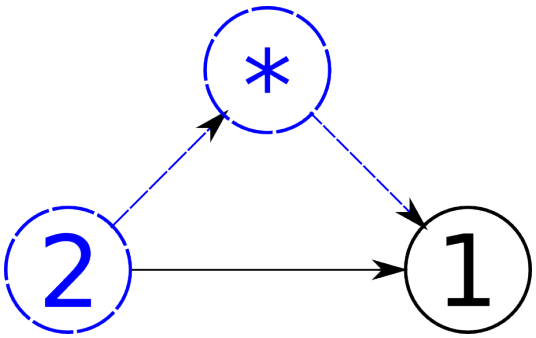
(b) …it has to ask for the cell state of its neighbors.



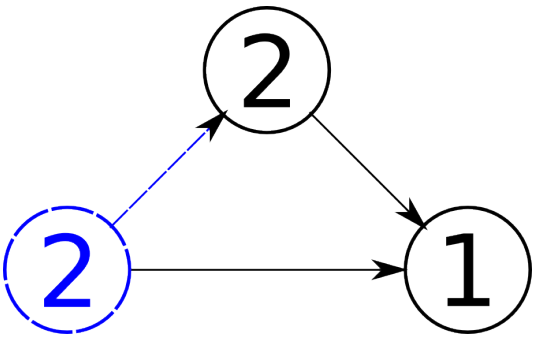
(c) The first neighbor has no continuations on its own so his cell state is one.



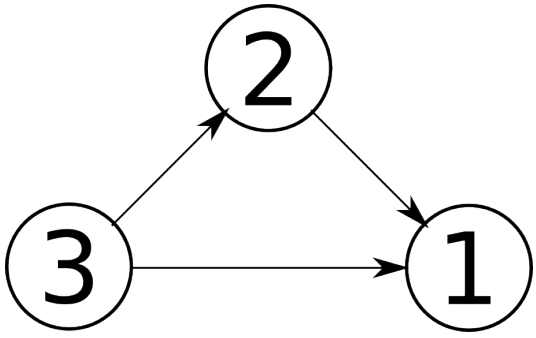
(d) It signals back one to the first cell, which keeps it plus one as the preliminary highest state.



(e) The second neighbor has no proper cell state. It has to figure it out by asking its own neighbors recursively.



(f) Its only neighbor has already a proper state and it can be taken immediately plus one as the cell state of the current cell.



(g) Receiving two as the cell state of the second neighbor the start cell takes the higher state plus one as its final state.

Figure 4.1: Algorithm steps for the unweighted tracking cellular automaton on a simple graph.

Often vertices have been identified with plain hits. In this analogy the graph would be a subset of the neighborhood of sense wires. In this thesis we often use neighborhood as synonym to the graph edges. If we want to stress the acyclic requirement we are using forward neighborhood as an equivalent term meaning the particles flight as advancing forward in the edge direction in the graph.

4.3 Weighted algorithm

As it will be discussed later in the concrete description of the tracking algorithm plain hits are not sufficient cells to base the cellular automaton on. In the unweighted algorithm all cells and edges are equal among each other, which is an unrealistic simplification, if we used cells, which have more substructure to them than the plain hits. The forward connections between them could be expressed with a refined quality measure as well. So we introduce the weighted longest path algorithm as a full replacement to the traditional cellular automaton used in tracking, which has not been done before.

In the weighted version each cell is assigned a weight, which represents an amount of points to be gained by picking it up. The points shall be a summable measure of quality. For track parts this could be their number of contained hits as a first approach to their true informational content. Moreover a weight gets assigned to all edges as well. The weight of the edges has to be compatibly summable to the weights of the cells. It can also be chosen negative for this purpose. For track parts this could be the overlap between two of them, if there happens to be one. In this case the weight would circumvent the double counting of the hits, if the two track parts were in the same path.

The modified algorithm is carried out similarly to the original one, but the assignment of the states is altered as follows. The state to be assigned to the cell S_i is its weight W_i plus the maximum of the neighboring cell states S_j combined with the weight w_{ij} of the edge connecting to them according to

$$S_i = W_i + \max_j(w_{ij} + S_j) \tag{4.2}$$

The edge weight matrix w_{ij} is most likely sparse. Edges that are not present in the graph should be absent values in the matrix. The construction of the longest paths is carried out as before by starting from the cell with the highest state and following the edge responsible for its high state combing edge weight and cell state of the neighboring cells again.

The weighted cellular automaton is a true generalization of the unweighted form and we can specialize to the unweighted algorithm by assigning a weight of one to all cells

and a weight of zero to all available edges.

$$W_i = 1$$
$$w_{ij} = \begin{cases} 0 & \text{for available edges} \\ NaN & \text{for unavailable edges} \end{cases} \tag{4.3}$$

Chapter 5

The local tracking algorithm

> If everything when it occupies an equal space is at rest, and if that which is in locomotion is always occupying such a space at any moment, the flying arrow is therefore motionless.
>
> Arrow paradox — Zeno

In this chapter we outline the local tracking algorithm developed during this thesis using the ingredients described before.

As we develop a local tracking algorithm, we want to rely only on local features of the central drift chamber. Hence we are not using the interaction point as a reference to distinguish inside from outside. Also we are not introducing any distinction between clockwise and counterclockwise inside the central drift chamber. As a result we have to postpone the decision, if a found segment is actually co-aligned with the particle track or if it presents the reverse travel direction of the particle, to a rather late stage of the tracking process.

The increased beam background from the high luminosity of the accelerator creates new challenges to the tracking algorithm. We expect that the drift time, used during the tracking procedure, can reveal, if a particular hit is made by background and is therefore unrelated to a close by trajectory. The active use of the drift is one of the main differences to the algorithms used in the former Belle experiment.

The tracking shall only figure out subsets of hits that belong to the same particle and specify an ordering in which the hits occurred. The final fit of the track parameters are to be made in a later stage by the Genfit algorithm based on the Kalman filter. However, we can provide initial parameters needed for the Kalman filter from the fast linear least square fits used in tracking.

5.1 Overview

To since the tracking procedure is rather involved we outline the main steps before we describe them in detail. We treat the tracking as a two stage process:

1. We seek to construct small track segments limited by the superlayer bounds from elementary wire hits in a two dimensional tracking procedure.

2. We combine segments to three dimensional track candidates.

Building segments The first part receives the raw hit information and aggregates them segments in the following steps

1. Translate the raw data and combined it with the detector geometry.
2. Group the hits into *clusters*.
3. For each cluster:
   a) Build triples of wire hits, called *facets, as cells* to be given to the cellular automaton.
   b) Construct the graph edges by searching *neighbors* of each *facet*.
   c) Retrieve the *paths* from the *cellular automaton* in a *multi-pass* manner.
   d) Reduce the *paths* of facets *to segments*.
   e) Make the *two possible orientations* of the segments available to the track building stage.

Building tracks The second part accepts the segments from the former stage and combines them with roughly the same principle despite of that no clustering has to be performed.

1. Build *triples of segments as cells* to be given to the cellular automaton.

2. Construct the graph edges by searching *neighbors* of each *segment triple*.

3. Retrieve the *paths* from the *cellular automaton* in a *multi-pass* manner.

4. Reduce the *paths* of segment triples to three dimensional *tracks*.

5. Decide whether the tracks should interpreted as *reversed*.

6. *Export to* track candidates, which can be fitted by *Genfit* algorithm.

We proceed by describing each step and justify the structure of the algorithm in more detail.

5.2 Building segments

Starting from bare wire hits we seek to them combine to reasonable segments. We understand segments as parts of tracks that are contained within a superlayer of the central drift chamber. Since we want to use the cellular automaton technique to construct the segments as maximal paths of a directed graph, we have to figure out, how to translate the event information as closely as possible to a graph.

At the beginning of the tracking process we have no z–coordinate information available. So we start tracking in two dimensions using the reference position of the wires as position of the hits. A z–displacement away from the reference rotates all stereo hit positions around the z–axes. However the relative coordinates between hits of the same track should not change significantly.

Also the drift length has to be taken at the reference position as a preliminary estimate. Care has to be taken for the unknown in–wire propagation time of the signal as well as the time of flight delay. Both additional times exhibit a common offset to the drift radii of neighboring hits. Despite of this difficulty we cautiously use the drift length information.

5.2.1 Clusterization

As a first step we realize that wire hits are grouped together in bunches or clusters. The clusters of the event previously shown in Figure 1.6 are color coded in Figure 5.1. They are naturally formed by the superlayer boundaries as well as by distantly separated tracks. A forward neighborhood for the cellular automaton should not span from one cluster to the next, since the gap from one to the other is too large to relate them in this manner. So instead of composing one graph, which has multiple unconnected subgraphs, we build clusters of wire hits first and construct more interconnected graphs on them.

As a benefitial side effect of the clusterization we can reduce unstructured scattered background by discarding clusters with a small number of hits. The beam induced background is expected to produce such small clusters, because it is mainly caused by low-energy particles traveling parallel to the beam line and the sense wires.

The clusters can be generated from the single wire hits by a iterative expansion of a symmetric relation on them. Suitable are all symmetric subsets of the closest and secondary neighborhoods of sense wires as presented in Section 1.2.7. We utilize the closest neighborhood, but may want to expand it further, if we assume that some hits along a track are not recorded and we want to bridge gaps along them. The segments we are about to construct will be fully contained within a certain cluster and consequently in a single superlayer.

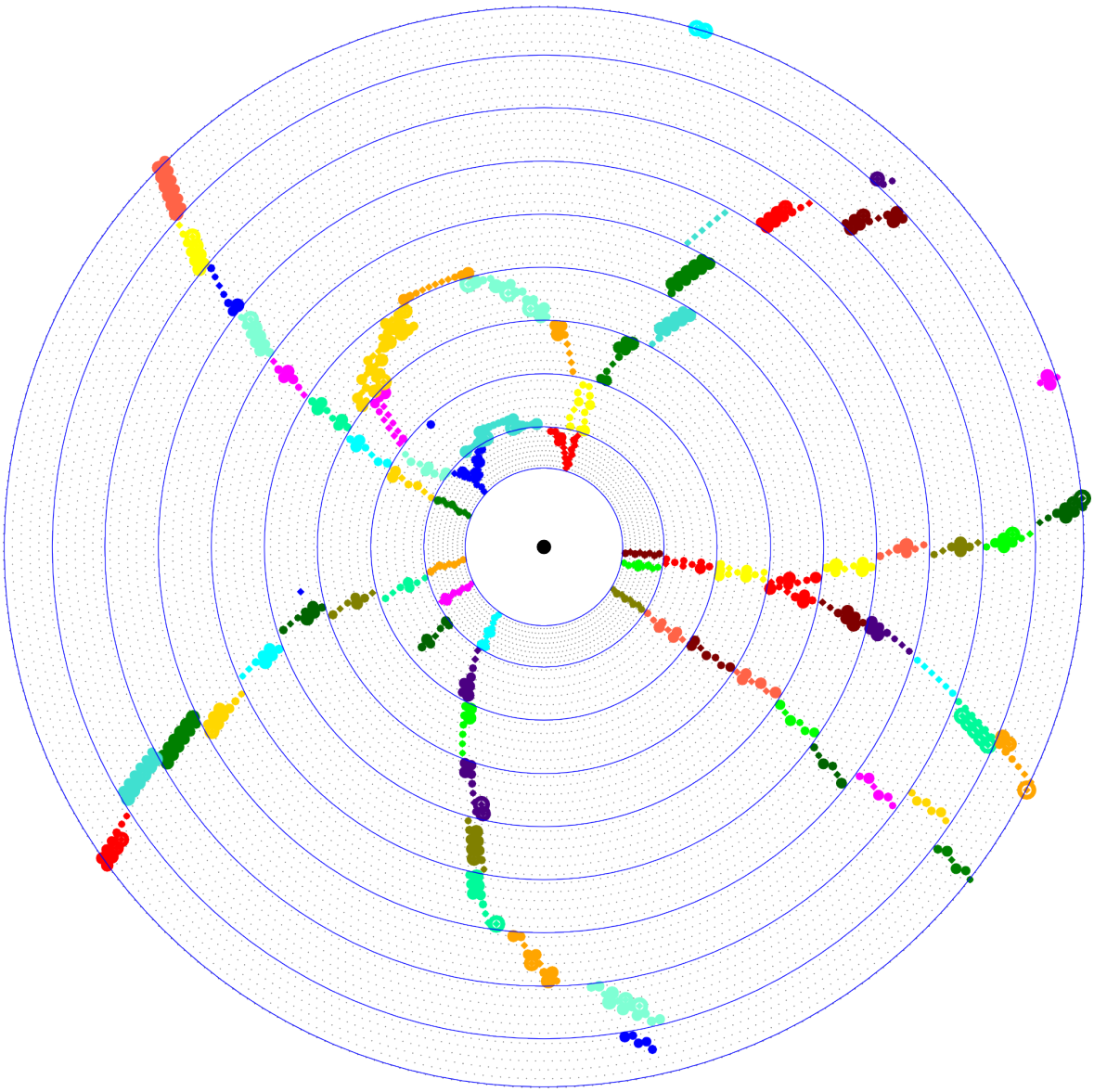
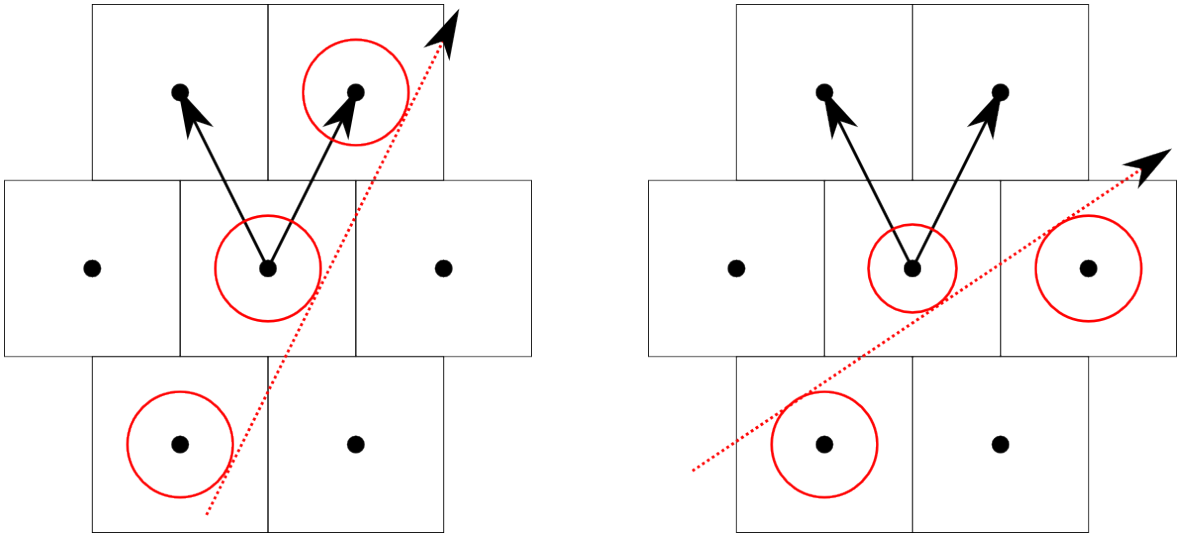


Figure 5.1: The already shown typical event in the central drift chamber with the hit clusters highlighted with a different color based on the internal cluster number (compare Figure 1.6). The clusters were obtained by repeated expansion of the closest wire hit neighborhood.

5.2.2 Determination of automaton cells and neighborhood

Wire hits as cells

One approach, that has been brought up for the Belle II central drift chamber, is to use the hits themselves as cells [16]. But single hits do not carry any direction of flight information to determine the following hit. Hence we had to rely on the static neighborhood of sense wires, which is narrowed even further by the acyclic graph requirement. Also we do not want to introduce a distinction between counterclockwise and clockwise into the graph. The only reasonable neighborhood left contains the one and eleven o'clock neighbors as is shown in Figure 5.2. Because this neighborhood is directed outwards, it can only reflect particle movements from the interaction point leaving the central drift chamber. However like Figure 5.2b illustrates, particles with slightly lower momenta or even curlers cannot be tracked with it.



(a) High momentum particles are trackable

(b) Lower momentum particles are not trackable

Figure 5.2: An acyclic neighborhood of single wire hits pointing away from the interaction point. Possible neighbors of the central wire hit are marked with the black arrows. The actual particle track is indicated by the red arrow.

Moreover, this kind of neighborhood can only pick up a single hit for each layer, while in reality two hits of a single particle in one layer are well possible. A second hit in one layer significantly narrows the region through which a particle passed and provides valuable information to the fit stage. Using this inadequate neighborhood we had to correct for this loss by manually adding hits from the same layer[16]. To conclude plain wire hits do not constitute suitable cells for the cellular automaton.

Tangents as cells

As we saw in Section 2.2.4 we can construct tangents to drift circles. Indeed tangents reflect the desired positional information on the trajectory and represent a direction of travel. To demonstrate a few concepts also relevant to the final choice of the automaton cell we discuss their features here in more detail as well as their disadvantages.

To retain locality we should base the tangents on closest neighboring hits. For a pair of hits there are four possible tangents (see Figure 5.3). So a proposal for the automaton cell has to include the right–left passage information for each of the two wire hits in order to acquire a unique travel direction. By taking all closest neighbors into account, the tangents are not limited to particles moving strictly outwards, but can reflect sideways travel and even inwards travel as it might happen for curling particles.

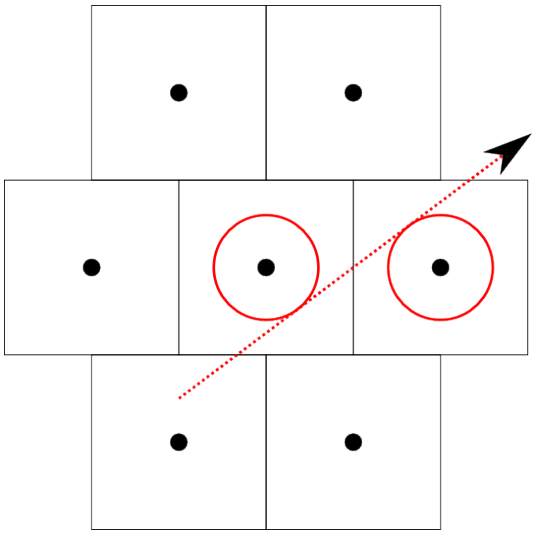
The forward neighbors of a tangent are easily found to be the tangents that start on the wire hit, where the first tangent ended. The right–left passage information on this wire hit shall match. The neighborhood can be refined by cutting on the angle between the direction of the first tangent to the neighboring one. The cut should be relatively narrow, since we do not expect a large curvature on the scale of single drift cells for trackable particle trajectories. We also want to keep the number of valid neighbors reasonably low.

This argument for the use of tangents as the automaton cell seems flawless but for a single detail. The radius of the drift circle, the tangents are based on, is not well known. By considering Equation (2.42) the in–wire–delay and the travel time of the particle can influence the relevant angle linearly for crossing tangents. In fact the error in the estimated direction of the tangent can be as large as the narrow cut, which we have to apply to avoid too many edges in the graph. To get an impression of the magnitude of the deviation, if the two true drift circles were just slightly larger, consider the Figure 5.4. Note that for passing tangents the travel direction is not changed, because the two mentioned delays cancel during the calculation of κ in Equation (2.45).

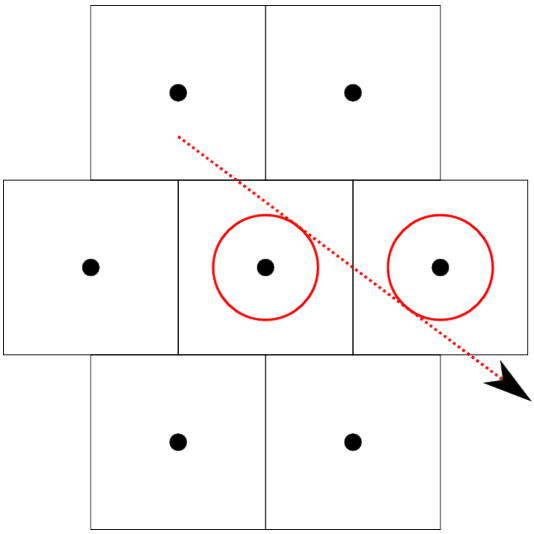
An alternative explanation of this disadvantage is that two uncertain support points, which are located close to each other, present an unstable connecting line. If a particle traveling outwards is hitting two wires in the same layer the two touch points to the drift circle lie relatively close together. So the unstable tangent to the same layer neighbor hit may not be considered as a graph edge, which either prevents the second hit in this layer to be included in the path or even break the segment at this point. Both alternatives are not acceptable.

Facets as cells

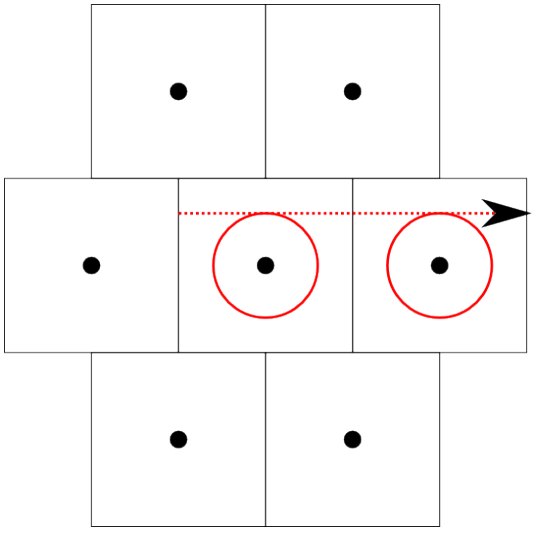
In order to overcome the uncertainty in the tangent direction we consider triples of wire hits as the bases of the automaton cell. The three wire hits shall have an order of their assumed occurrence along the track. So we have a start, a middle and an end wire hit.



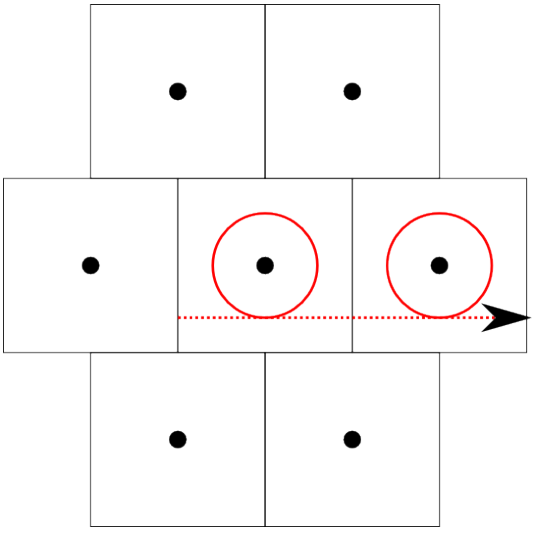
(a) Crossing tangent with right–left passage $\alpha_A = -1$ and $\alpha_B = 1$



(b) Crossing tangent with right–left passage $\alpha_A = 1$ and $\alpha_B = -1$



(c) Passing tangent with right–left passage $\alpha_A = 1$ and $\alpha_B = 1$



(d) Passing tangent with right–left passage $\alpha_A = -1$ and $\alpha_B = -1$

Figure 5.3: Four tangents to a pair of drift circles. Each of the four displayed tangents should be an automaton cell. The left hit is always assumed to occur prior to the right on. The arrows mark the tangent direction. Four additional tangents will be constructed for the reversed travel direction hypothesis.

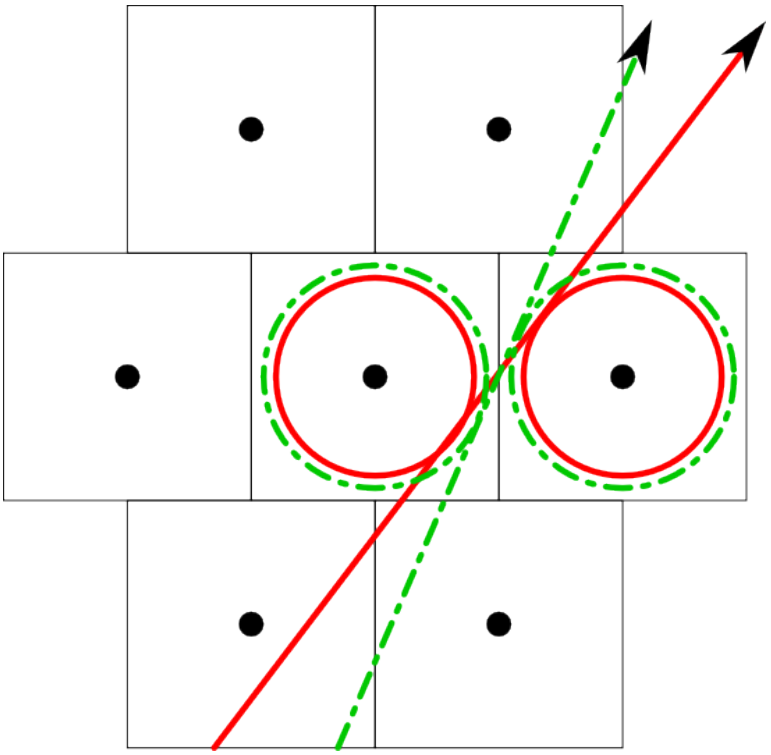


Figure 5.4: Crossing drift tangents are sensitive to a common delay time. Here the true dotted green drift circles are slightly larger then the measured red ones resulting in a very different reconstructed tangent direction. A neighboring tangent might be subjected to an error of equal magnitude. In compensation we had to widen the cut on tangent directions to neighboring tangents, which will lead to an unfeasible amount of false connections.

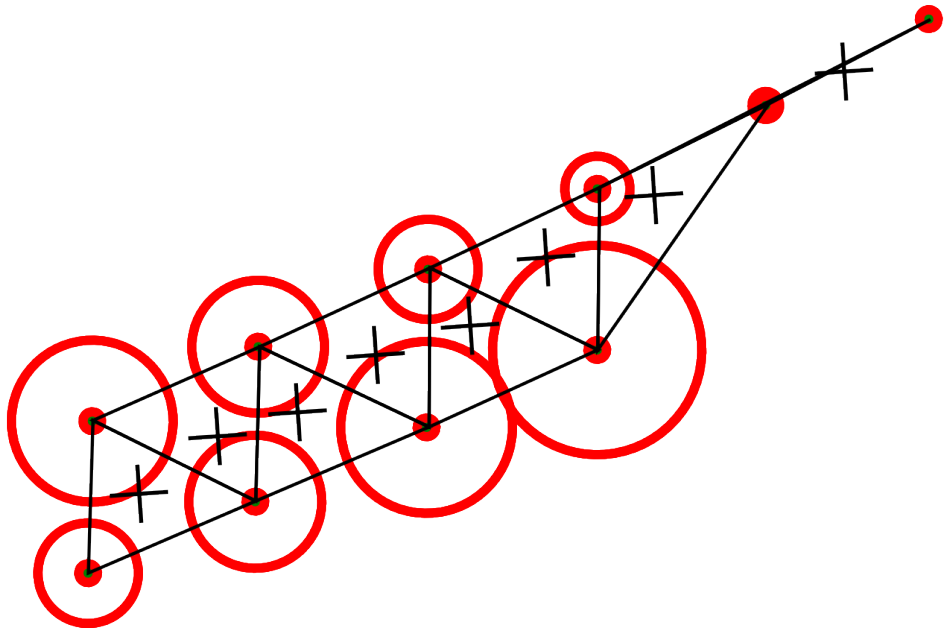
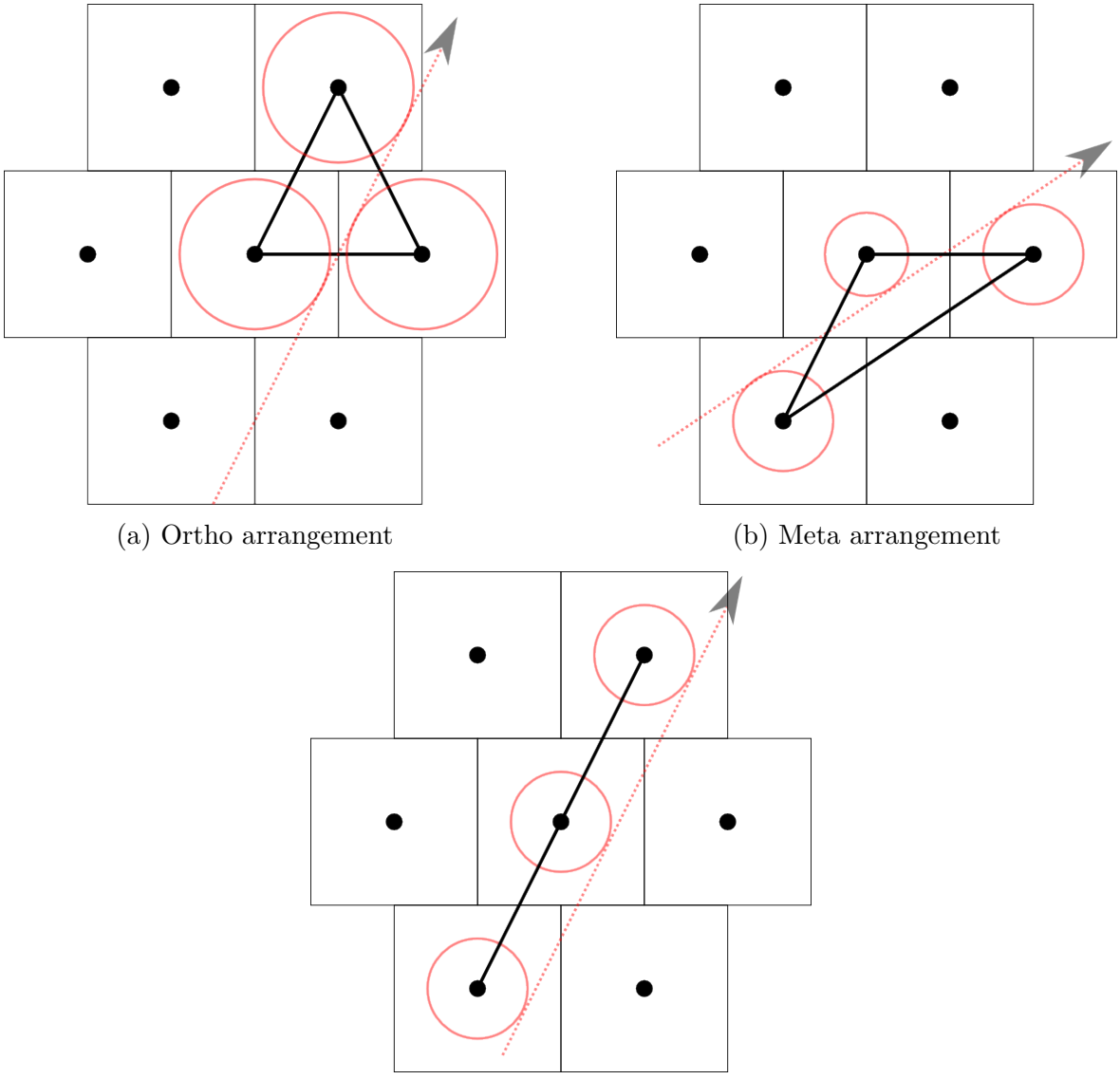


Figure 5.5: Segment of a simulated particle track in the central drift chamber. The particle originates from the interaction point located to the left. Hits from the same layer are vertically aligned. Facets triangulate the true particle position (black crosses) in the general case (part to the left), however some degenerated cases have to be taken into account to cover all possibilities (part to the right).

The start wire hit and the middle wire hit should be closest neighbors as should the middle and the end wire hit. Similarly to the tangents we have to add the right–left passage information to each of the three hits in order to constrain to a unique direction of travel. To give a more handy name to these ordered triples with attached right–left information we refer to them as facets. Again we use the full closest neighborhood of wires allowing for all travel directions independent of the location of the interaction point.

From Figure 5.5 we can learn, that in the common case facets can frame the likely positions of a particle desirably. For each layer a facet can naturally pick up two hits, which overcomes a main disadvantage of the other two cell candidates. But we also have to take triples into account which do not form triangles, especially in case particles have only one hit in each layer. As shown in Figure 5.6 we distinguish three general shapes of triples up to rotations and mirror symmetry.

Construction of facets Since the facets have a rather rich substructure not all possible combinations of three neighboring wire hits with attached right–left passage information may yield valid facets. So during the construction of these automaton cells we



(a) Ortho arrangement

(b) Meta arrangement

(c) Para arrangement

Figure 5.6: The three different shapes of facets up to rotations and mirror symmetry. The nomenclature is taken from the bimethylated benzene molecules called xylene, which have the same three possible orientation in their methyl groups.

have to check their compatibility.

The correct alignment of start, middle and end wire hit to the direction of flight is asserted by comparing the direction of the three tangents which can be constructed from the three drift circles. The facet is accepted, if the angular deviation among the three tangent lines is sufficiently small.

We seek to keep the number of constructed facets as low as possible to avoid superfluous computations, while maintaining a high efficiency of probable facets. So the correct determination of compatibility is the most crucial decision to be optimized.

Construction of the neighborhood Whether two facets are considered forward neighbors can be decided by their properties by the following requirements.

1. A possible forward neighbor has its start based in the middle wire hit of the original facet.
2. Also the neighbors middle shall be identical with the end wire hit of the facet.
3. The right–left passage information shall match on the two wire hits common to both facets.
4. The end of the neighbor should be different from the start of the original facet to deny a reverse in direction. (This requirement is not necessary as we want to check the flight direction as well, but it prevents some unnecessary calculations of the latter.)
5. The angular deviation of the direction of flight from one facet to the other has to be small.

In contrast to the single tangent the whole facet has an associated flight direction derived as follows. Besides corner cases with zero drift length there are only two general passage possibilities for the three tangents:

- All three are passing tangents, in case all three wire hits have the same passage information.
- One of the tangents is passing, the other two are crossing, in case one of the wires has a different passage information then the other two.

In the first case we are not affected by the delay time error. In the second case the two crossing tangents are affected, but the error can be diminished by a fit to the three drift circles and the travel direction becomes a stable parameter, to which we can apply a tighter cut.

Moreover, because of the three point wire hits in the facets, they may reflect the curvature of the particle trajectory, which might make a cut on it applicable. This value is susceptible to the delay time error though and has not been used to date.

Using facets overcomes all mentioned disadvantages of the lower order cell constructs. We consider the graph of the facets as most optimal representation of the event topology, because we put as much information into the vertices by the hit alignment, as we are using for the edges from the travel directions. Adding a fourth wire hit will only increase the amount of cells to be generated, while they have a large redundant overlap among each other. Higher order cells would increase the computation time, while adding only little accuracy.

Weights in the cellular automaton

Having found the best cell we assume the cell and edge weights. The default weight of a facet is three, since it carries three wire hits equating three pieces of information. The default weight of an edge is minus two, because the overlap between adjacent facets in two wire hits would lead to the double count of the latter.

Both measures can be refined by a probability, that the three wire hits make a facet and the quality of connection between two neighboring facets. The framework for this enhancement will be provided, but to figure out the correct measures to be inserted is beyond the scope of this thesis.

5.2.3 Application of the cellular automaton

The derived facet cells and their forward neighborhood can be given to the weighted cellular automaton. After we determined the longest path, we pick up all facets along it to construct a first segment. Since we are considering one cluster at a time constructing a single segment from it is most often enough. But if there were close-by tracks in one cluster, more segments should be constructed.

One way to resolve this is to remember all possible starts of locally maximal paths in the graph and construct a segment by beginning from each of them following the highest states. A starting point of a path is each vertex which has no ancestors. While traversing the graph during the cellular automaton, we can already mark these vertices without an additional traversal by assigning a flag to them. The advantage in building many segments is the high efficiency since the true segments are most likely among the constructed ones. On the downside we are generating a lot of potentially overlapping segments if the graph had converging branches near the beginning of the path. Subsequently we have to apply a best subset analysis on the generated segments.

Another approach is to apply the cellular automaton several times and construct a single longest segment in each pass. In between the passes facets, that have wire hits in common with the used ones, are blocked and the cellular automaton will ignore them during the next passes. In this scenario additional segments are only built if the remaining longest path still has a specified minimum of hits. Therefore we can avoid to build segments from a single facet, if we considered that three aligned facets

are probably not made by a true track. The limit can be set lower than three, which means that single facet tracks are built as well.

Initially the first procedure had been implemented, but was replaced with the second method for their speed advantages. The code of the first implementation is still available, if it turns out to be useful again.

5.2.4 Reduction to segments

Finally, we obtained mutually exclusive paths of facets, which we want to simplify to segments being a sequence of reconstructed hits. Each reconstructed hit is assigned a right–left passage information and a probable reconstructed position on the drift circle, where the primary ionization occurred. Both quantities are averages calculated from the involved facets as follows.

The paths generated by the cellular automaton contain facets of three wire hits. Because of the overlapping neighboring facets every wire hit in the segment is included up to three times. We reduced this multiplicity be averaging out the information presented by these three instances to a single reconstructed hit. Within a facet each hit has two candidate points for a reconstructed ionization position being the two touch points of the two tangents, which are based on this hit. So in total we have up to six candidate points for the reconstructed position and we take the average of them to be stored as the position estimate. The average point is most likely not exactly on the drift circle and we scale its displacement from the wire reference such that it lies on the drift circle again. The latter should only be a minor correction and could be neglected. Also, the two dimensional reconstructed hit is assigned the right–left passage information identical in all facets. The reconstructed segments of the already shown typical event are plotted color coded in Figure 5.7

5.2.5 Remarks on orientation

As the construction of facets and their neighborhoods takes the full wire hit neighborhood into account, they have no sense for inside or outside. Also the facets cannot reveal the travel direction of the particle, because the loss of momentum in the forward direction is not visible in three wire hits only. Hence for each facet, there will be a twin facet corresponding to the reversed travel of the particle. With the same argument each twin will have a reversed neighborhood attached to it. This behavior is quite reasonable, since it truly states our absence of knowledge at this point.

Hence during the generation of the segments both directions of travel are available in the graph with the same score, since the reversed segments consist of the reversed facets and neighborhood relations contributing the same weights as their unreversed counterparts. Because after the blocking procedure or the best subset analysis the

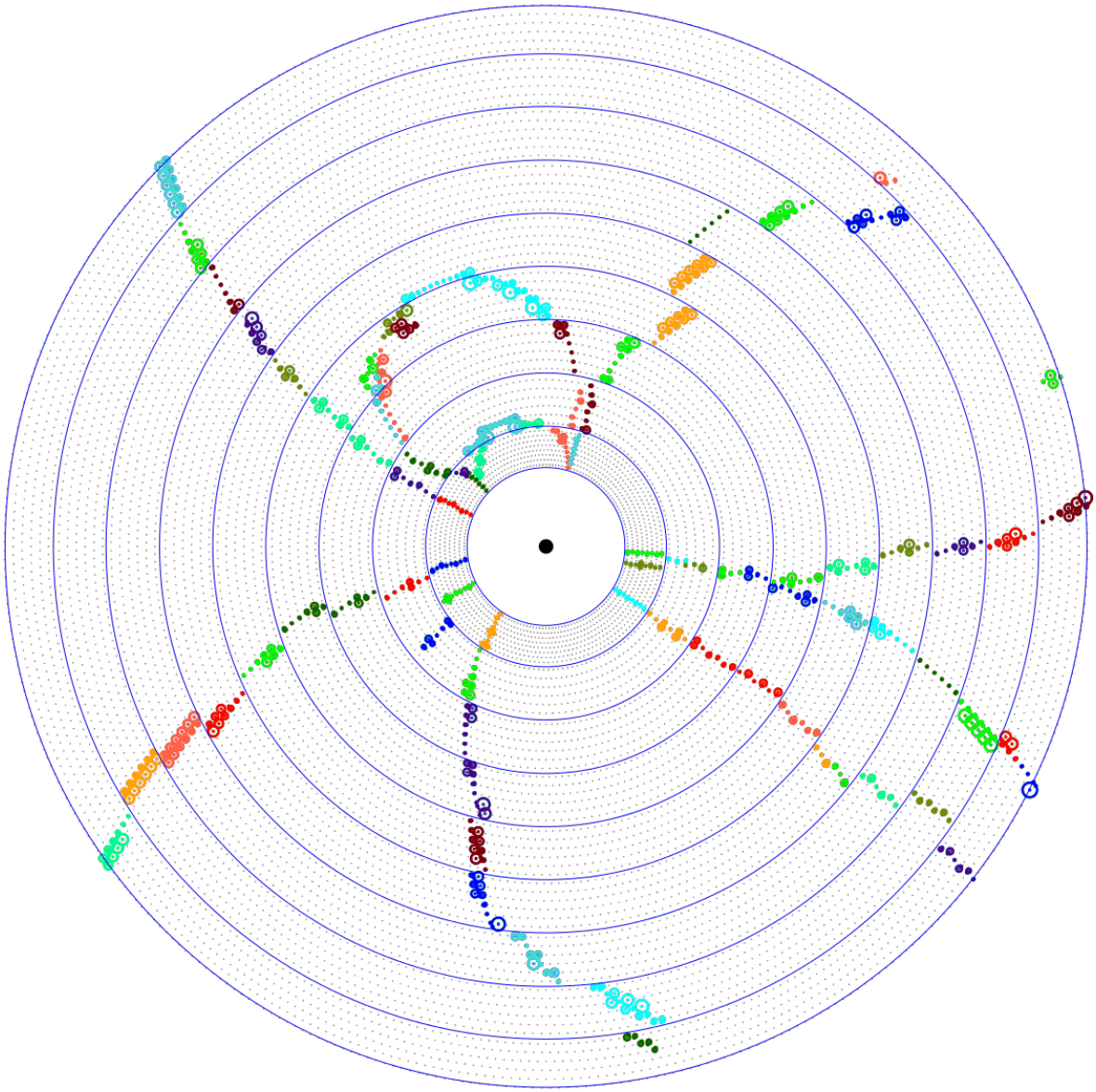


Figure 5.7: A general event after the segment construction phase. The differently colored wire hit groups represent the segments which have been created. Each segment has a color assigned based on their order of creation. Most of the clusters contain a single segment, only a few consist of more. Some segments are split, where there should only be one, due to the imperfections in the facet creation cuts.

segments are mutually exclusive, only one of the two orientations prevails. Which one gets chosen is basically random.

By examination of the built segments we cannot decide the correct travel direction either, because they are still to short to figure out the direction of decreasing momentum. We have to make sure, that the output of the segment building stage also reflects this limited knowledge properly. We manually add the reversed version of each segment, in order to pass both valid possibilities to the track building stage.

On the down side double the amount of all facets, neighborhood relations and final segments are created, then it would be necessary. To avoid this there had to be severe organizational changes in the generation of facets and their treatment, which has not been done for the simplicity of the implementation. Still it might offer some optimization potential.

5.3 Building tracks

Similarly to the building of segments from facets we proceed to combine tracks from segments as their basic building blocks. Pursuing the local tracking technique, continuations of segments are sought in neighboring superlayers. Still the combinatorics in the possible segment combinations is high and we rely on the cellular automation to generate long tracks from local connections. The efficiency of the algorithm is defined by our sophisticated choice of the cell to use in the automaton as well as their forward neighborhood.

5.3.1 Fitting segments

As we seek to combine segments from different superlayers, we require some sort of extrapolation to judge the compatibility of the segments and the found continuations. Since the fits used should not be too time consuming, the already mentioned fast linear least square methods will provide the preliminary trajectory estimates.

Two dimensional trajectories

The received segments from the former stage are long enough to make stable trajectory fits to them. However the estimates are only sensible for segments located in axial superlayers. During the development process we noticed that fitting to stereo segments often gives the wrong sign curvature. In the stereo superlayers the perceived particle track at the wire references position with a high momentum z–component is increasingly rotated as it travels forward. The increase in the rotation can be large such that a fit misidentifies a counterclockwise traveling particle as clockwise or vice

versa. Consequently, we refrain from fitting to stereo segments and only obtain two dimensional trajectories for the axial segments.

The fits are carried out with the linear least square of planes to the projected paraboloid presented in Section 3.2.3. We intentionally transported the right–left passage information with the reconstructed hits in the segments to enhance the fit with the drift length information. Due to their short circle arc segments with only a few hits might acquiring a too high curvature from a fit using only their drift circle. In this situation the reconstructed ionization position can be utilized as additional points in the fit to generate more straight trajectories.

Our example event is shown in Figure 5.8 with fitted two dimensional circle trajectories to all identified axial segments. The figure is only meant to give an overview of the general event complexity in terms of the fits. To understand the quality of the found trajectories, the detailed view in Figure 5.9 illustrates that the fits follow the true trajectory reasonably well. By using the fit we can easily find the location of the next segment along the track.

SZ- trajectories

Having a proper two dimensional fit circle we can reconstruct the z–coordinate of stereo hits associated with it, using the procedure described in Section 2.3.5. As each reconstructed hit in the segments has an attached two dimensional position, we use it to assume the skew line to be substituted into Equation (2.102). Like the whole wire is related to its reference position, the reconstructed two dimensional position represents many possible three dimensional positions. Similarly to the wire, all of these three dimensional hit positions are assumed to lie on a skew line, which can be obtained by moving the wire line by the displacement of the hit position. During the reconstruction of the z–coordinate the two dimensional hit position is drawn onto the circle trajectory giving a likely three dimensional position of the primary ionization (see Figure 5.10). We also estimate the travel distances in the transverse plane s for each hit in the segment as the arc length on the circle relative to the first hit in the segment.

Given the transverse travel distance s and the z–coordinate, we are able to fit a sz–line to the stereo segment with the normal linear least square method. Hereafter, the sz–line can be used to judge the compatibility of two stereo segments. Note that the stereo segments on their own are of little use, because they do neither provide a valid circle nor z– or s–information on their own. They always need an associated axial segment and its fitted trajectory to reveal their information.

5.3.2 Determination of automaton cells and neighborhood

In analogy to the segment building stage, we discuss the different possible choices for the cellular automaton cell.

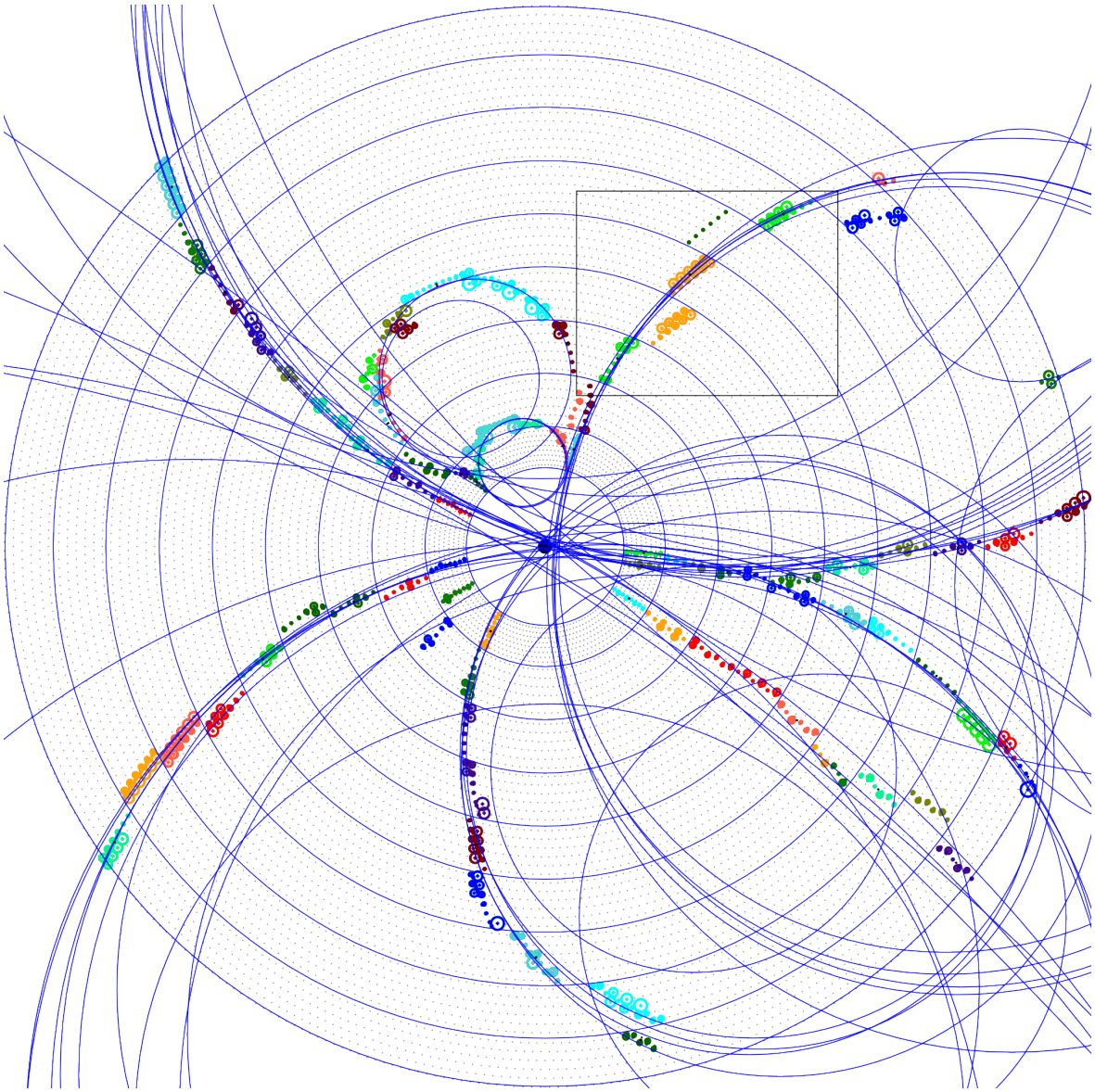


Figure 5.8: Example event with fitted circles for each of the axial segments. The display is quite complex to give an impression of the possible combinatorial effort, that has to made to disentangle it. The black rectangular region is plotted again in Figure 5.9 to show the actual fit quality in more detail.

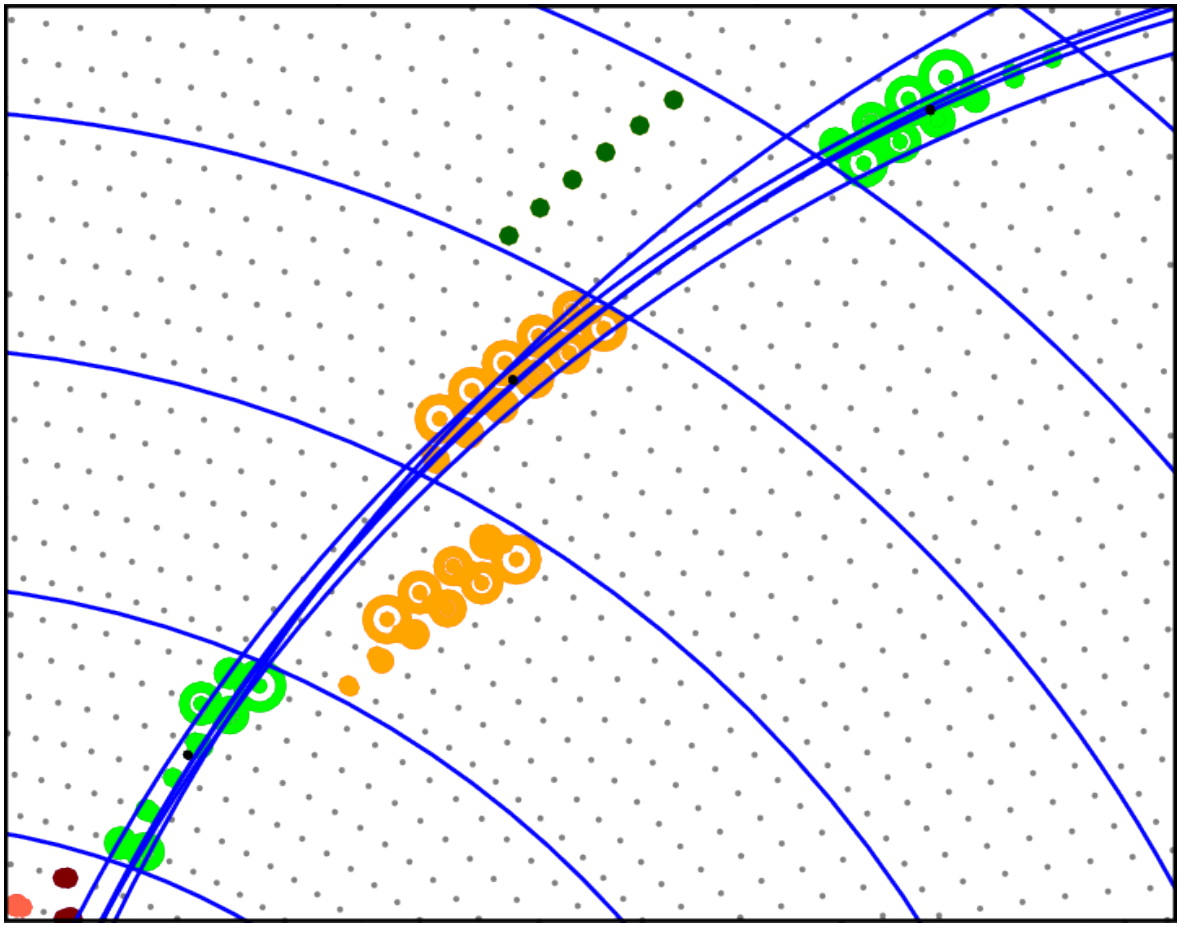


Figure 5.9: Detailed event view with three axial segments and two stereo segments in adjacent superlayers. The axial segments were fitted with the parabolic Riemann fit technique. The resulting circles are shown in blue. The true trajectories are reflected very well by the circles and the correspondence between the segments can easily be drawn. Note that there are five circles in this figure, because the two axial just outside this view are plotted as well.

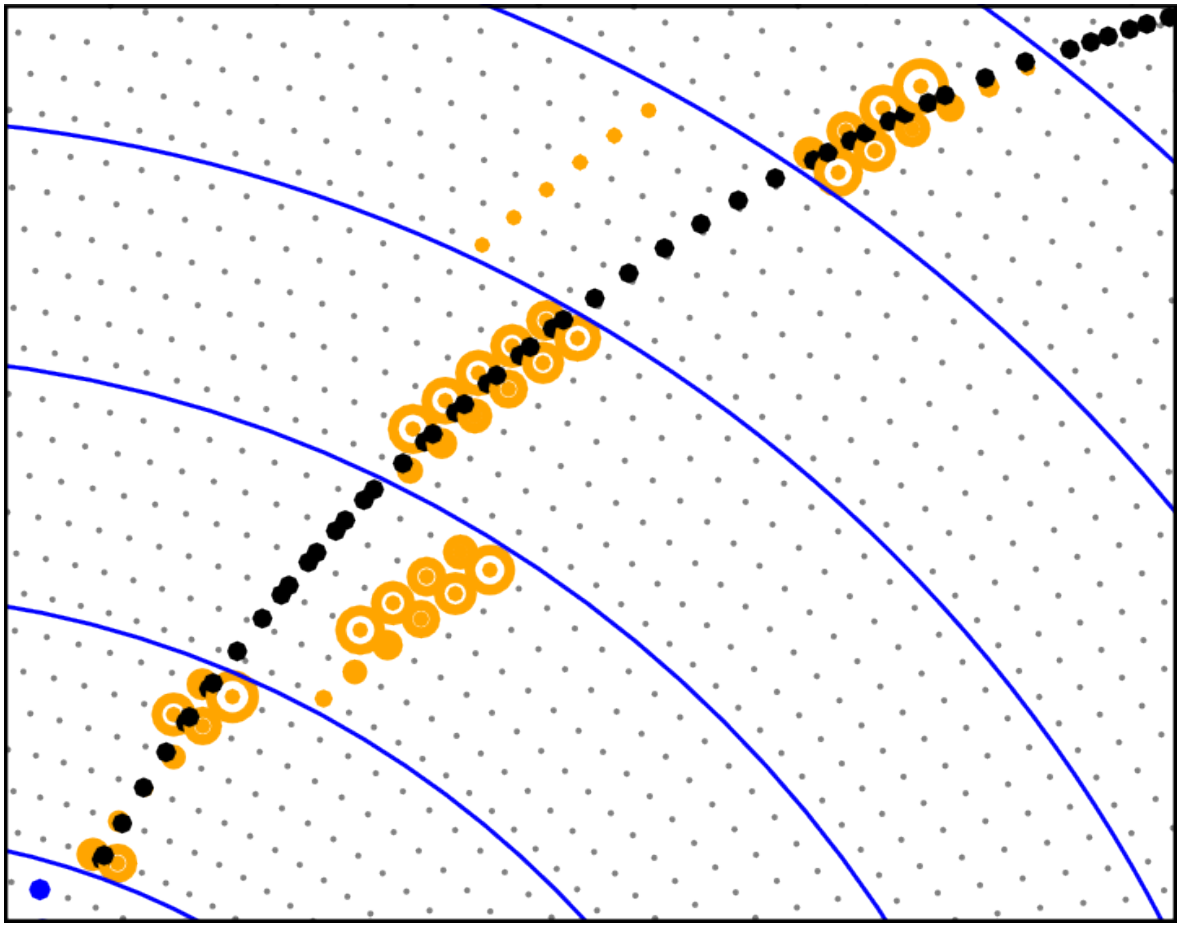


Figure 5.10: The detailed view of the example event showing the reconstructed xy position of the segments (black dots). The trajectory used for the reconstruction is the fit to the axial segment in the middle. The reconstructed points of the stereo hits seem to be quite far off the reference position, because the track is in the far backward region of the central drift chamber, where the rotation of the stereo layers compared to their reference position is strongest.

Single segments as cells

Single segments are not suitable, because they have no attached z–coordinate on their own. In a scenario, where we want to generate a neighborhood of single segments by connecting axial superlayers to the adjacent stereo layers or vise versa, we can reconstruct the z–coordinate of the stereo segment only in the context of neighboring pairs. During the cellular automation this information from a neighbor pair cannot be transported to a subsequent neighbor pair. Hence a stereo segment can be interpreted as lying in the forward region of the central drift chamber in conjunction with one axial segment and as lying backward, if combined with another, where both connections can end up in the same cellular automaton path. Corollary the cellular automaton path would not make sensible tracks regarding the z–coordinate.

Pairs of segments as cells

Using pairs of segments combining one axial segment and a stereo segment from adjacent superlayers introduces z information attached the cell.

However, the pairs are asymmetric, which needs not to be harmful, but may place the following undesired property to the pair. Because the circle fit is based on the axial segment only, the stereo hits would be reconstruct on a part of the circle which is rather far away from its supporting points. Small errors in the two dimensional position map to huge uncertainties in the z–coordinate, because of the small wire skew. So the computed z–information on the side of the stereo segment far away from the axial segment is rather unstable and we risk forming wrong tracks regarding the z–component.

Triples of segments as cells

The uncertainty in the z–coordinate is effectively diminished by placing the stereo segments inbetween two axial segment. So we consider triples as cells, where the superlayer combination is axial, stereo and finally axial. Figure 5.11 gives an instructive picture of a segment triple also showing a circle fit to both axial segments. The two axial segments in the cell can be combined in one circle fit with small uncertainty, which gives a most reliable trajectory for the z reconstruction of the stereo segment in the middle, and a stable z–slope over transverse travel distance can be computed. Combining a two dimensional circle trajectory and an sz–line from the segment triple defines a full helix trajectory making the triple of segments a minimal reasonably fittable track. The three segments in the triple shall be ordered as they are assumed to occur in the track and we will refer to them as start, middle and end segment, where the middle is always a stereo segment. Triples of segments are considered the optimal choice, because more segments provide little additional information while increasing the unnecessary overlap between cells.

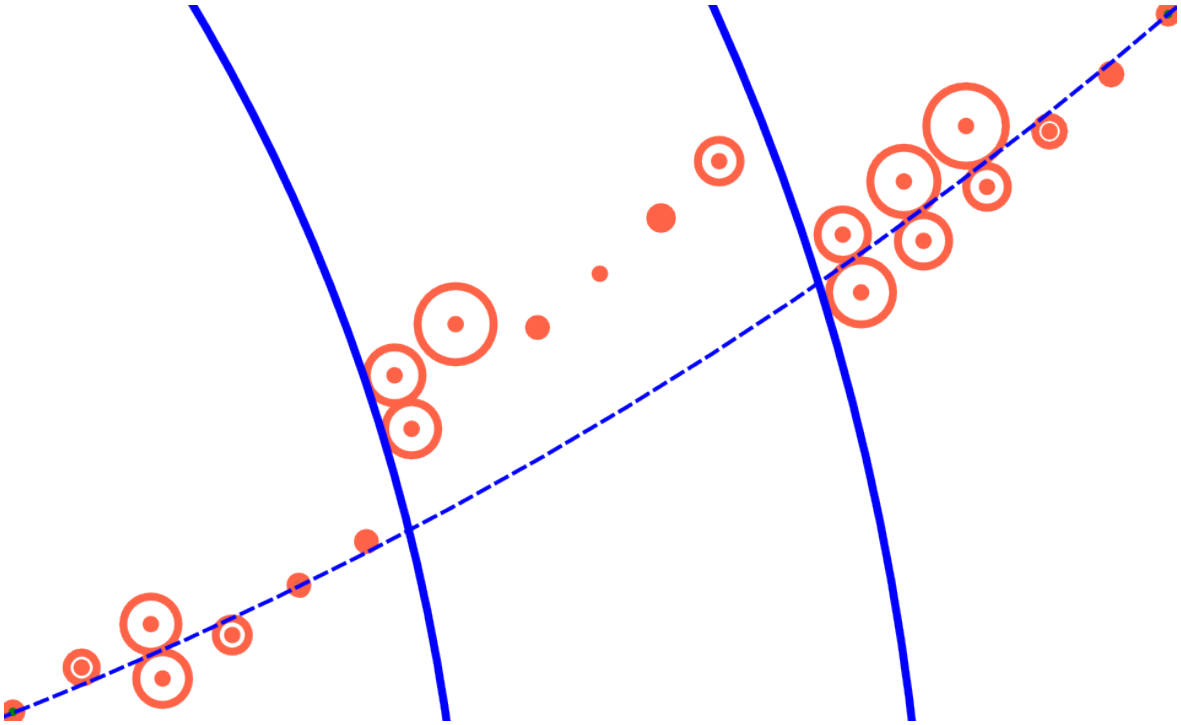


Figure 5.11: Segment triple with two axial segments and a stereo segment in the middle including a circle fitted to the two axial segments.

Construction of the cells Beginning with one axial segment taken as start, we first try to find a suitable axial end segment. The end axial segments shall be sought in the axial superlayer outside of the start superlayer, in the same axial superlayer or in the axial superlayer to the inside. We include the same superlayer as search option, in case we are tracking a curler which travels into the next stereo superlayer, but returns to the same axial superlayer without touching a different axial superlayer inbetween.

To decide the compatibility of the two axial segments, we check if the end segment lies forward in the direction of travel of the start segment and vise versa. The fits of the two axial segments are subsequently compared in their curvature and travel direction. As well we could fit both together in a single fit and investigate the overall χ^2–value. The distinction, whether the axial to axial continuation is correct, can be optimized further, since there are plenty of options, which have not been fully investigated yet.

Having a pair of axial segments, we attach a common circle fit to them and search for a stereo segment in the middle. Since we know the maximal rotation of the stereo layers in the middle, we can limit the region of the stereo superlayer from where we accept a segment. Since it should only prevent stereo segments too far away, for instance on the opposite side of the central drift chamber, to be considered, this acceptance region is slightly widened to compensate inaccuracies in the circle fit. We continue by checking the correct alignment of the stereo segment with the outer two axial segment. Subsequently the z–coordinates of the stereo hits in the segment are reconstructed using the circle formed by the two axial segment. If the reconstructed hit positions are suggested to lie outside of the central drift chamber, the stereo segment is rejected as a possible middle. This should not be made too strict since the z–coordinate is very sensitive to small errors in the fit circle. Finally and most importantly the sz–linearity of the reconstructed stereo segment should be investigated. The exact procedure has yet to be fixed, since there are many possibilities. .

Construction of the neighborhood The neighborhood of segment triples can be derived as follows. The start axial segment of the neighboring triple should be identical with the end segment of the original segment triple. Hence, two neighboring segments have an overlap in one axial segment. This neighborhood can be narrowed down by comparing the circle fit and sz–fits of the segment triples. Also the decision at this point is critical for the efficiency and it has to be studied in detail, because it is not obvious, what to do best.

Weights in the cellular automaton

Having decided the graph structure the weights to the cellular automaton have to be fixed. As default value the segment triples have weights equal to the total number of hits in the three segments. The edge default weights should again prevent the double counting of hits in the track. Therefore the negative of the amount of hits in the axial segment common to both segment triples should be taken as their weight.

These simple measures can be made more sophisticated by diminishing them with the connection quality inside the triples and among them. The cellular automaton makes sure, that the most valuable paths are built, so we could also accept kinks in the segment chain by giving them a low value.

We could also introduce a forward backward distinction into the graph, if we find that a certain connection is more favorable in one direction than into the other. For instance, if the momentum estimated in one triple is slightly higher than in the neighboring the forward connection can acquire a higher value then the reversed connection. It has to be investigated if such a distinction can be made at this level of detail. An asymmetry has not introduced yet and forward and backward are still indistinguishable in the graph.

5.3.3 Application of the cellular automaton

Similar to the segment building stage we can choose between a multi-pass application of the cellular automaton building one path at a time or a single pass generating all maximal tracks. Because a best subset analysis had to be made for the many tracks constructed in the second scheme, we choose again the multi-pass method for their simplicity. Inbetween passes all triples containing common segments to the built path are blocked.

As a result we obtain mutually exclusive sequences of segment triples. The smallest track that can be constructed in this manner has at least three segments and start and ends in an axial superlayer.

5.3.4 Reduction to tracks

From the segment triple paths we single out the hits mentioned twice in order to obtain a sequence hits. We save the right–left passage information for each of them in case Genfit might become able to use it. Additionally we keep averaged information of the z–coordinates and the transverse travel distance s shifted appropriately to be measured from the start of the track.

We also hand over the two dimensional fit as well as the sz–fit of the first segment triple to the track to provide initial values for the track fit carried out by Genfit. As the track might be interpreted as reversed to the true trajectory the fit of the last segment triple is stored in the track as well. If we want to reversed version to be fitted by Genfit the latter will provide the initial values for the Kalman filter.

5.3.5 Loose ends

Tracks that only contain a single axial segment cannot be obtained from the cellular automaton since the paths consist of at least one segment triple. Therefore particle with a large boost along the beam–axis would be lost, if they leave the central drift chamber at a small polar radius. To save these tracks we introduce single segment tracks in a post-processing stage after the automation, where we consider segments from the innermost superlayer which have more than eight hits. This is of course yet another decision to be investigated for their correctness.

Because the built tracks always end in axial layers, there are most certainly left over stereo segments, which should eventually get associated to a track. A procedure for this has yet to be invented.

5.3.6 Deciding the orientation issue

In the end we still have to fix the orientation of the tracks. If we are not introducing a forward backward distinction into the graph, the same argument applies as to the built segments and the tracks are generated in a random orientation.

We are forced to correct the tracks to the better orientation at this point. For now we decide for the tracks to move away from the interaction point. The best way how to break the forward–backward symmetry has yet to be determined. Possibilities are to reconstruct the decay tree from the interaction point during in this stage or leave it to Genfit to figure out the forward direction of the track, since it might be sensitive to the direction of decreasing momentum.

Chapter 6
Implementation

In this chapter we summarizes the context and organization of the concrete implementation of the local tracking algorithm. It only outlines the general purpose of the various used classes, since a full description cannot be given within the scope of this writing. A more detailed documentation of the implemented classes and methods is available from the author by request.

6.1 Framework

The whole implementation resides in the Belle II analysis software framework (basf2 for short), which handles all computations related to the Belle II detector from the data acquisition or the detector simulation up to the physics analysis. Figure 6.1 illustrates the general control flow of the framework, which is described in the following.

The Belle II analysis software framework mainly consists of combinable modules representing district steps in the analysis chain, which can be conveniently arranged in so called steering files written in Python [20]. It processes events one after the other handing the control from one module to the next in order specified in the steering scripts. In order to utilize the full capacity of modern multicore CPU hardware and their threading support, many module chains processing different events may run in parallel

The modules themselves can be written in C++ or in Python. Since compiled C++ code runs much faster, it is used for modules processing the numerous events. Python modules are useful to monitor the data and algorithms in the debugging stage, because of its variety general purpose tools and their ease of use.

Modules communicate their data through the data store facility. Because the data store shall be writable to disk the Belle II analysis software framework incorporates ROOT for data serialization. The ROOT data framework developed at CERN is the de facto standard for data handling in high energy physics, because it is able to serialize almost arbitrary C++ objects. For its mechanism to work for a specific class, it needs to inherit from ROOT's general base class `TObject` as well as it needs to acquire a method and data member dictionary. The latter has to be prepared by the CINT

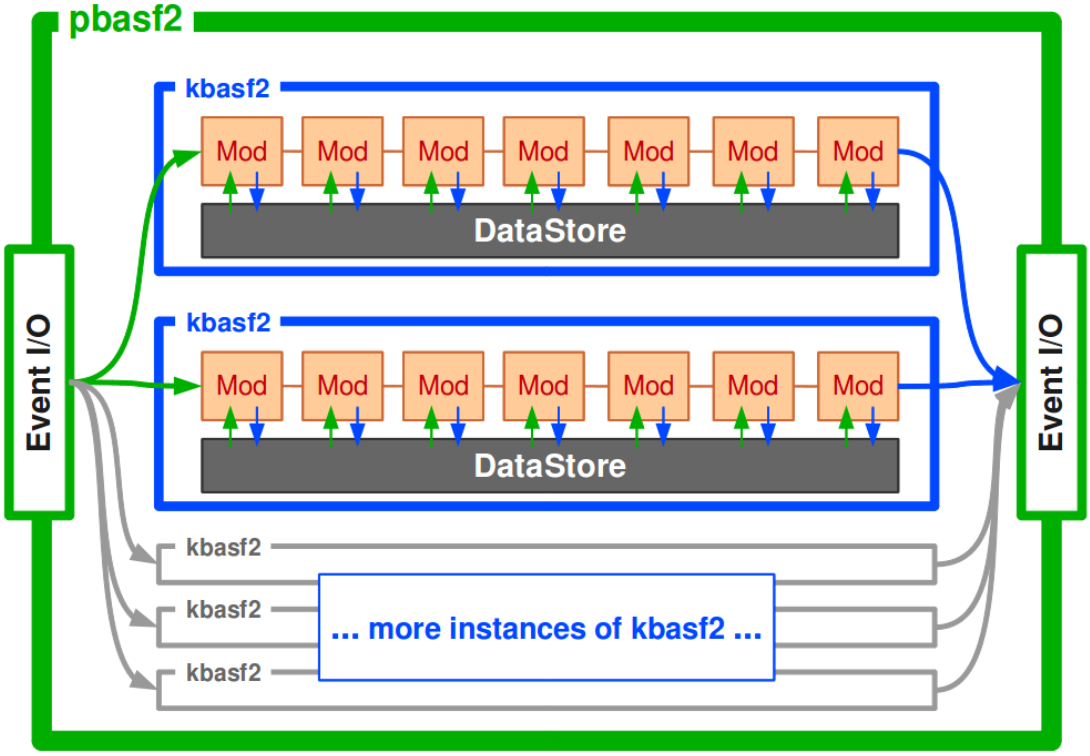


Figure 6.1: Scheme of the control flow of basf2 showing multiple executions of the same module chain processing events in parallel. Communication between modules is realized via the data store facility [20].

C++ interpreter of ROOT, which also puts limitations to the objects to be serialized. Especially the support for C++ template classes is rather rudimentary.

6.2 Input and output of the tracking module

The tracking is meant to run on the digitized data output of the central drift chamber as a module to the Belle II analysis software framework. Since the detector has not been built yet we rely on data from a Geant4 simulation and a subsequent simulated digitizer to mimic the response of the readout electronics to physics events as closely as possible, in order to run the reconstruction code such as tracking.

From the data store we receive the raw readout data via an array of `CDCHits`. Especially we require the TDC count of the drift time measurement, which we have to translate to a drift length first. The latter can be done by a calibration function provided by the framework (compare Figure 1.4). From the database presented by the framework we also obtain the current wire geometry to be associated to the hits.

The output of the tracking module shall be an array of `GFTrackCands` written to the data store, from where it can be fetched by the Genfit module for the final determination of the track parameters.

6.3 Mocking the TObject inheritance

The easiest way evaluate the correctness of the many steps in the tracking algorithm is to store intermediate objects to the data store and retrieve them from a Python module to generate plots and displays. All event displays in this writing have been produced in this manner from simulated physics events.

The inheritance from ROOT, which is mandatory for storable objects, puts a performance loss to every object construction, because of the virtual constructor of the `TObject` and its additional memory consumption. During local tracking we potentially create a lot of objects to gather the refined information as we combine more and more hits. For each higher order object, we want to write to the data store, every contained lower level object has to inherit from `TObject` as well. Hence the used memory attributed to `TObject` piles up from abstraction layer to abstraction layer.

Despite of that we decide to keep all tracking data accessible via the data store to Python during debugging, but seek to optimize out the inheritance to `TObject` for the actual run. We can implement a simple switch between debugging and optimized run by a single limited range compiler macro and the virtue of the empty base class optimization.

```
///// mockroot.h :

// Load the TObject.
#include <TObject.h>

// Define an empty object class.
class MockTObject { ; };

// Comment to deactivate ROOT support in local tracking.
#define USE_ROOT_IN_CDCLOCALTRACKING

#ifdef USE_ROOT_IN_CDCLOCALTRACKING
  //In case we want to use the intermediate data store output:

  // Use TObject as base class.
  typedef TObject UsedTObject;

  // Link the used macros for dictionary creation to the ROOT Versions.
  #define ClassDefInCDCLocalTracking(ClassName,ClassVersion)
     ClassDef(ClassName, ClassVersion);
  #define ClassImpInCDCLocalTracking(ClassName) ClassImp(ClassName);

#else
  //In case we do not use the intermediate data store output:

  // Use the empty mocking class as base class.
  typedef MockTObject UsedTObject;

  // Link the used macros for dictionary creation to empty macros.
  #define ClassDefInCDCLocalTracking(ClassName,ClassVersion)
  #define ClassImpInCDCLocalTracking(ClassName)

#endif
```

Listing 6.1: Definitions to inject the ROOT `TObject` base class into the tracking objects.

```
///// CDCWire.h :

// Include the definitions.
#include <mockroot.h>

// Inherit from UsedTObject.
class CDCWire : public UsedTObject {

  CDCWire()  ///< Default constructor for ROOT compatibility
  ~CDCWire() ///< Empty deconstructor

  //....

  /// ROOT macro to make a dictionary declaration for CDCWire.
  ClassDefInCDCLocalTracking(CDCWire, 1);

}; //class


///// CDCWire.cc :

#include "CDCWire.h"

/// ROOT macro to make a dictionary implementation for CDCWire.
ClassImpInCDCLocalTracking(CDCWire)

CDCWire::CDCWire() {;}
CDCWire::~CDCWire() {;}
//....
```

Listing 6.2: Example definition of a injectable object subjected to the TObject mocking paradigm.

The Listing 6.1 shows the realization of the paradigm for mocking the `TObject` inheritance. In the classes we want to eventually put on the data store we inherit from `UsedTObject`. A minimal definition of such a class is given in Listing 6.2. By commenting the macro `#define USE_ROOT_IN_CDCLOCALTRACKING` from Listing 6.1 we can deactivate the `TObject` inheritance of all objects inheriting from `UsedTObject` at a central location.

Class	Bytes
UsedTObject	16
Vector2D	24
Vector3D	48
Line2D	48
ParameterLine2D	64
GeneralizedCircle	56
BoundSkewLine	80
CDCTrajectory2D	96
CDCTrajectorySZ	64
CDCWire	136
CDCWireHit	56
CDCRecoHit2D	56
CDCRecoHit3D	80
CDCRecoTangent	104
CDCRecoFacet	272

(a) Memory with `TObject` as base class.

Class	Bytes
UsedTObject	1
Vector2D	8
Vector3D	16
Line2D	12
ParameterLine2D	20
GeneralizedCircle	16
BoundSkewLine	28
CDCTrajectory2D	28
CDCTrajectorySZ	16
CDCWire	80
CDCWireHit	40
CDCRecoHit2D	24
CDCRecoHit3D	32
CDCRecoTangent	40
CDCRecoFacet	128

(b) Memory with empty class as base class.

Table 6.1: Memory demand of the most fundamental tracking objects with and without the inheritance to `TObject`. By switching of the inheritance we can substantially diminish the memory, which also reduces the run time of the tracking.

The impact on the memory consumption is given in Table 6.1. It demonstrates that by deactivating the `TObject` inheritance, we can save considerably more than half of the memory needed by the most fundamental tracking objects. This reduction also affects in the time to run the tracking, which is reduced by up to 20% depending on the concrete composition of the algorithms.

6.4 Overview of implemented classes

6.4.1 Data structures

This section describes the implemented objects to store geometrical and event related data. All of them can be written to the data store by the inheritance from `UsedTObject`. The only exception to this rule is the `WeightedNeighborhood`, which is a template that was not accepted for serialization by the ROOT.

Graph for the cellular automaton

`AutomatonCell` represents a vertex of a graph for the cellular automaton. It is meant to be a base class to all objects used as cells. The instances store the state S_i to be filled during automation, the weight of the cell W_i and various status flags, which are employed to mark, if the cell

- has already been set during the automation,
- has ancestors, hence is not a start point of locally maximal path,
- has been traversed in current recursion, which helps to encounter a cycle in the graph and/or
- was blocked by a former pass of the cellular automaton.

`WeightedNeighborhood<class AutomatonCell>` stores the edges of the graph for the cellular automaton. It is essentially a sparse matrix of weights w_{ij} implemented based on a C++ `std::multimap`. The indices are pointers to the `AutomatonCell` objects, which implies that the storage location of the referenced vertex objects should not be altered and the same object is needed for retrieval of its neighbors as it has been used during the creation of the edges. Since we need different types of neighborhoods, a neighborhood of wire hits for the clusterization as well as neighborhoods of facets and segment triples for the cellular automation, we implement the `WeightedNeighborhood` as a template instantiable with the different objects to which it shall relate.

Geometrical objects

This section briefly outlines the implementation of the concepts presented in Chapter 2. Since we are working on a rather framework–like implementation, the geometry objects provide more methods than are essentially needed for the local tracking algorithm. For instance we give conformal map methods in case another algorithm wants to build upon them.

We are not basing the geometrical objects on the vector implementations `TVector2` and `TVector3` of ROOT, because of their inconvenient interface and methods. Also they

always have the performance penalty from the `TObject`, which cannot be switched off. Still we want to keep the already discussed interface to Python, which also excludes the commonly used `HepVectors` from our pool of choices. Hence we introduce our own implementation of fundamental two and three dimensional vectors with the `Vector2D` and `Vector3D` class and built all other geometrical objects upon them. In this manner we are also able to define orientational properties consistently as we wish and can introduce specialized methods, which may give boosted performance.

`Vector2D` represents an ordinary two dimensional vector. In addition to expected vector methods like dot product, cross product, the polar radius and angle etc. the class implements methods regarding

- the orientation of the xy–plane with the desired definitions of right and left,
- the translation to local coordinate systems by calculating the parallel and orthogonal components relative to a coordinate system vector and
- the conformal map.

`Vector3D` provides a basic three dimensional vector. The interface of the `Vector3D` is similar the `Vector2D`, but does not include the orientational features of the latter. Since we are often projecting to the xy–plane by neglecting the third component of the `Vector3D`, we avoid the construction of a new `Vector2D` object from the first two coordinates in our custom implementation. By basing the `Vector3D` on a `Vector2D` member variable we can return a reference the latter in order to perform the xy–projection instead of a copy.

`Line2D` implements the normal line representation. It serves as a base of the sz–trajectory. The class supports

- the calculation of signed distance to the line,
- the assignment of points to the right or left half plane,
- the mapping from on coordinate axis to the other,
- the determination the closest approach to a point and
- the computation of intersection point of two lines.

Also the line can be moved active and passively parallel to the coordinate axis.

`ParameterLine2D` represents a parametric line, which will be used by the tangents to the drift circles to encode the direction as well as their touch points to the drift circle. The interface is most similar to the `Line2D`, but some methods regarding the parameter of the line have been added.

`GeneralizedCircle` provides the concept of the generalized circle. It supports the calculations discussed in Section 2.3.2, which include

- the signed distances to the circle,
- calculations of the travel distance between two points,
- points of closest approach and
- the conformal map.

`BoundSkewLine` implements the line representation for the sense wires. It can be constructed from two end points and provides

- the point of closest approach to the z axis as the reference $\vec{W}$ for the two dimensional tracking,
- the wire skew σ and
- z–coordinates of the two end points in order to test, whether points are still inside the central drift chamber.

It also helps the reconstruction of the z–coordinate of the hits in conjuncture with a generalized circle.

Topology of wires

The sense wires, layers and superlayers are implemented as the `CDCWire`, `CDCWireLayer` and `CDCWireSuperlayer`, while the `CDCWireTopology` class provides the memory for each kind. The whole topology has to be initialized before the first event is tracked and after each change in the detector geometry. Each of the four classes provides an interface to obtain their instances and their is no need to construct them during the actual tracking procedure.

`CDCWireTopology` is meant to be a singleton class and keeps wires, layers and superlayers in `std::vectors` sorted for their respective ordering scheme. All valid instances can be retrieved by indexed lookup accepting all possible id combinations (compare Section 1.2.6).

Since the wire neighborhoods are bound to their superlayer, the logic for the neighborhoods should be implemented in the `CDCWireSuperlayer`. The `CDCWireTopology` only provides an interface to obtain the closest and secondary neighbors of an arbitrary wire by delegation to the appropriate `CDCWireSuperlayer`.

CDCWires models the physical sense wire by the skew line representation. It is responsible for unpacking the geometry data presented in the database of the framework to the `BoundSkewLine`, from which we can also obtain the axial or stereo type of the wire.

The `CDCWire` may store additional precomputed data related to a sense wire. Since the wires only need to be initialized once before the start of the tracking the time spent in their creation is relatively unimportant. To date we only keep the range of stereo twist of the wire as an additional variable. Precomputed parameters of the facets are another possible information that could be stored, but this has not been done yet.

The wires are sortable as a triple of superlayer id, layer id and wire id in this order, or equivalently by their encoded wire id. Also there is a distinct lowest wire with the encoded wire id 0, which is employed in the lookup of compound hit information building upon the sense wires.

CDCWireLayer represents a range of `CDCWires` by storing a pair `iterators` to the `std::vector` in the `CDCWireTopology`. Similarly to the wires, the `CDCWireLayer` may provide additional precomputed variables for the tracking. For now the average of the following wire parameters are stored:

- Minimal polar radius
- Average skew
- Average forward and backward z coordinates
- Average forward and backward polar radii
- Average angular twist of the layer
- Numbering shift relative to the first layer in this superlayer

The last property may need some explanation. Because of the hexagonal wire arrangement, it is conventional, whether the numbering of wires is shifted clockwise or counterclockwise, when crossing layer boundaries from one to the next. However, the latter is important in order to identify the wire ids of closest neighbors. It is during the construction of the `CDCWireSuperLayer`, because this specific property can only be determined by comparing the numbering of one layer to the next and not from within the wire layer.

CDCWireSuperlayer describes a range of `CDCWireLayer` with the same stereo type. Therefore it keeps two `iterators` into the storing `std::vector` of `CDCWireLayers` in the `CDCWireTopology`.

The retrieval of closest and secondary neighbors resides in the `CDCWireSuperLayer`, which implements the lookup by manipulation of the wire ids. Additionally it can compute the neighborhood relation from one wire to the next in terms of the o'clock position, which is useful to determine the shape of a facet or the general direction of a tangent.

Trajectories

Based in the geometrical objects we define trajectories, which add a physical meaning to the abstract constructs described before.

`CDCTrajectory2D` represents a circular trajectory in xy–projection of the central drift chamber based on the `GeneralizedCircle`. In addition it marks a certain point on the circle as start point of the transverse travel distance scale. It helps

- the setting and retrieval of the transverse momentum,
- the extrapolation in the transverse plane.,
- the calculation of the transverse travel distance from the start point and
- the adjustment of the start point of the trajectory.

The latter returns the shift in the transverse travel distance that has to be subtracted from all measures taken before the adjustment. This value can be used to synchronously move an associated sz–trajectory to the new start point .

`CDCTrajectorySZ` holds a linear fit of the transverse travel distance over the z–coordinate based on the `Line2D`. The start point of the trajectory is associated with the $s = 0$ point on the line. Additionally it provides

- the translation from transverse momentum to z–component of the momentum vector,
- the mapping functions from the transverse travel distance to the z–coordinate or vise versa and
- the shifting of the start point of the trajectory.

The latter can receive the travel distance shift from an associated `CDCTrajectory2D` in a consistent manner.

Hits and higher order constructs

This section describes the various classes aggregating local hit information for the algorithm. They contain up to three wire hits and are presented in the order of their first usage during the algorithm, hence in the order of more detailed reconstructed information.

All of them exhibit a particular total ordering scheme with a distinct lower bound to all valid objects of their respective kind, which makes faster lookups into sorted ranges of those objects applicable.

CDCWireHit is meant to attach the geometry information from a CDCWire to the CDCHits of the event given to the tracking algorithm. The CDCWireHit stores a pointer to the wire in CDCWireTopology, a pointer to the CDCHit as well as the index into the array on the data store for later translation to Genfit track candidates. Besides that the CDCWireHit stores the drift length, which is translated from the TDC count at this point.

CDCWireHits are sortable by their supporting wire to keep neighboring wire hits close together, which is necessary to rapidly find the closest neighbors from a sorted range of wire hits. Since we want to build clusters of the wire hits they inherit from AutomatonCell to store the cluster number as their cell state (see the Clusterizer below class).

CDCRecoTangent is a pair of CDCWireHits including a right–left passage information for both of them. It uses a ParameterLine2D to represent the direction of the tangent and the two touch points to the drift circles. The first touch point is located at parameter value of zero, the second at a parameter value of one.

The sorting scheme for the tangents is based on the following properties in the stated order:

1. Start wire hit
2. Start right–left passage information
3. End wire hit
4. End right–left passage information

In this manner all tangent starting on the same wire hit are arranged close to each other in a sorted sequence, which makes binary searches for them feasible.

CDCRecoFacet represents a triple of CDCWireHits including a right–left passage information for each of the hits. The three different tangent lines to the wire hits can be obtained from it to judge the connection to a neighboring facet. Also a possible reconstructed ionization position can be estimated for all three wire hits. CDCRecoFacet inherits from AutomatonCell, since it is used in the cellular automaton.

Also CDCRecoFacet have an associated total ordering, which compares the following properties:

1. Start wire hit
2. Start right–left passage information
3. Middle wire hit
4. Middle right–left passage information
5. End wire hit
6. End right–left passage information

Possible forward neighbors can be obtained as a range from sorted sequence of facets, since we are seeking a certain combination of start and middle wire hit including their right–left passage information (compare Section 5.2.2). The sorting scheme is essential to speed up the retrieval of neighbors for the generation of graph edges.

`CDCRecoHit2D` describes a likely position of the primary ionization in the transverse plane including a right–left passage information of the assumed track. It is the building block of the segments to be passed from the first to second stage of the tracking and it represents the point information used by the Riemann fit.

`CDCRecoHit3D` describes a likely three dimensional space point of the primary ionization in the central drift chamber also including a right–left passage information α and a transverse travel distance s. The `CDCRecoHit3D` also

- carries out the reconstruction of the z–coordinate of a stereo `CDCRecoHit2D` in conjuncture with a `CDCTrajectory2D`,
- represents a point to be used in a sz-fit and
- is the building block of the tracks, which are generated during the second stage of the tracking algorithm.

Collections of the hit classes

To provide a homogeneous interface to the various kinds of required hit sequences and sets, we implement them as generic collection templates. There are two template classes `CDCGenericHitVector` and `CDCGenericHitSet`, which are wrappers to the `std::vector` and `std::set` writable to the data store, that also provide homogeneous sorting and lookup member functions as well as specialized methods relating to their wire and hit content.

Because of the limited support of ROOT for template classes, they have to be instantiated in a rather unfamiliar way. Please consult the code documentation for more details. Classes building upon instances of these templates are stated in the following.

`CDCWireHitCluster` represents a group of close-by wire hits. Therefore it is an instance of `CDCGenericHitSet` template for pointers to `CDCWireHits`. The wire hits have to be kept in a different storage location, since the `CDCWireHitCluster` only describes the cluster by an aggregation of pointers.

`CDCRecoSegment2D` models a segment to be produced in the first stage of the tracking. The class is an instantiation of `CDCGenericHitVector` for `CDCRecoHits2D`. We also provide `CDCAxialRecoSegment2D` and `CDCStereoRecoSegment2D` as equivalent `typedef`s to document, which kind is expected in a certain piece of code.

`CDCSegmentTriple` is implementing the cell of the automaton in the second stage of the tracking. Besides pointers to the three segments the class contains a circle fit to the two axial segments and a sz-fit for the reconstructed middle stereo segment.

For fast lookups into sorted ranges `CDCSegmentTriple` are sortable for the pointer of the start segment. Again the sorting is crucial to find the suitable neighbors of segment triples during the building of their forward neighborhood.

`CDCTrack` uses the inheritance from the `CDCGenericHitVector` template in order to represent a sequence of `CDCRecoHit3D`. Due to multiple scattering, the accuracy of the simple circle and sz–fits is only given locally, which is why we attribute a total of four fits to the track:

- a circle fit to the start of the track,
- a sz–fit to the start of the track,
- a circle fit to the end of the track and
- a sz–fit to the end of the track.

The fits give the initial values to Genfit, where the later two indicate these values in case we interpret the track in the reversed direction.

6.4.2 Fitting

In the following we state the implementation of the fast fit models discussed in Chapter 3. For the necessary matrix calculations we use the handy Eigen matrix library [15]. Note that the two fitters have been made accessible from Python for direct checks of their correctness.

`CDCRiemannFitter` carries out the Riemann fit to segments by an singular value or eigenvalue decomposition. The class fits the `CDCTrajectory2D` to single segments or to pairs of segments using either

- only the drift circles with the right–left passage information,
- only the reconstructed positions or
- both of these information.

Despite of fitting to general circles the fitter can be constrained to fit circles through the origin.

`CDCSZFitter` can adjust the `CDCTrajectorySZ` to fit a three dimensional reconstructed stereo segment. It implements a normal line fit, which can either optimize the z–distance or the sz–distance of the line to the given points.

6.4.3 Generic algorithms

Cellular automaton

The generation of longest paths from a graph described in Chapter 4 can be split into two steps:

1. *A*ssignment of states and
2. *B*uilding of the paths

which we implement in different classes.

`CellularAutomaton<class AutomatonCell>` implements the recursive assignment of the cell states, ignoring already blocked cells. It concurrently marks the cells that have no ancestors using the status flags of the `AutomatonCell`. The reference to the cell with the absolute highest state is returned as a result of the cellular automation. Since the graph is represented by different types of concrete automaton cells and a `WeightedNeighborhood<>` of them, we need provide the concrete cell type as a template parameter.

`CellularPathFollower<class AutomatonCell>` After the states have been assigned, we pick up the path of highest cell states. The procedure is implemented in the `CellularPathFollower` class. It can either create the single highest path from the graph by starting from the cell with the absolute highest state returned from the `CellularAutomaton`, or create all locally maximal paths by starting from all vertices that have no ancestors. Again the template parameter to the `CellularPathFollower` is the concrete cell type used in the two stages of the tracking algorithm.

Clusterization

The concept of clustering is related to the cellular automation as it groups vertices from a symmetric graph. Hence, we employ the `WeightedNeighborhood` and the `AutomatonCell` in the clustering of wire hits.

`Clusterizer<class AutomatonCell>` constructs the clusters by iterative expansion of graph edges from a vertex that has not been assigned to a cluster yet. We can keep track of items that have already been assigned to a cluster by setting their cell state to the cluster number. Note that in the current implementation no recursive function calls are needed, which makes the implementation in `Clusterizer` class as efficient as possible.

To date the only concrete template parameter to the `Clusterizer` is the `CDCWireHit`, since we are only clustering wire hits. Nevertheless we keep the implementation general in case we want to cluster other entities such as the segment triples before proceeding to the cellular automation.

Creation of neighborhoods

For the cellular automaton as well as for the clusterization a `WeightedNeighborhood` filled with the edges of the graph to be traversed is required. If we had n randomly arranged vertices, we would have to check n^2 possible connection in order to single out the valid ones, which is too time consuming and hence renders to whole local tracking approach useless.

To diminish the number of considered connection the sorting scheme of the three relevant objects `CDCWireHit`, `CDCRecoFacet` and `CDCSegmentTriple` has been chosen to keep possible neighbors of a concrete instance in a continuous block within a sorted sequence. So by limiting our search of connections to this block of possible neighbors, we reduces the amount of connections to be checked to $n \cdot m$, where m is the number of expected possible neighbors. A small additional time of $\log n$ has to be spend to find the start of the neighbor block for each of the vertices. Still time consumption is cut to a feasible level since

$$n \cdot (\log n + m) \ll n^2 \tag{6.1}$$

`NeighborhoodBuilder<class NeighborChooser>` During the generation of neighborhoods we have to search the lower bound of the block of possible neighbors in the sorted sequence and consider each connection to the following instances in the sequence until we leave the block of possible neighbors. This general traversal logic of possible neighbors is carried out by the `NeighborhoodBuilder`, but since the details regarding,

1. where the lower bound of possible neighbors is located,
2. when the current block of possible neighbor is exhausted and
3. whether the possible neighbor is acceptable as well as which weight shall be assigned to the edge,

can vary for the different neighborhoods to be created, we factor them out into a `NeighborChooser` object as template parameter to the `NeighborhoodBuilder`. With this advantageous separation we are able to test the performance of different versions of the edge creation as well as support this step with Monte Carlo information to study the effect of the other tracking steps without imperfections in the graph creation. The concrete `NeighborChooser` interface used to obtain the answers to the three detailed questions is described in Section 6.4.5 below.

6.4.4 Creators

Each concrete step in the tracking outlined in Section 5.1, which is not covered by the generic algorithms, is implemented in a separate class. Because most of them are in charge of creating some sort of higher level object from lower order constituents, we are referring to them as creators.

Building of segments

`WireHitCreator` combines the hit information from `CDCHits` with the geometry information into a `std::vector` of `CDCWireHits`, which is sorted for the encoded wire id for later lookup.

`FacetCreator<class FacetFilter>` creates the `CDCRecoFacets` from the wire hits and their neighborhood. It generates all possible combinations of neighboring triples of wire hits including the right–left passage information, but delegates the decision, whether a facet should be constructed and which weight shall be assigned, to a different object introduced as the template parameter. Similar to the `NeighborhoodBuilder` this decision object has been factored to have distinct point for optimizations independent of the handling of the combinatorics. The interface of the `FacetFilter` is given in Section 6.4.5 below. The facets are sorted after their creation for the subsequent application of the `NeighborhoodBuilder`.

`SegmentCreator` carries out the reduction from `CDCRecoFacets` paths after the cellular automation to `CDCRecoSegments` by averaging out the hit information.

`SegmentReverser` creates a reversed version of every given segment to properly reflect this ambiguity in the track building stage.

Building of tracks

`SegmentTripleCreator<class SegmentTripleFilter>` composes the triples of segment ordered for their sorting scheme from the given segments implementing the procedure described in Section 5.3.2. As the `FacetCreator` it delegates the decisions,

- whether an axial to axial segment combination is acceptable,
- if an axial–stereo–axial segment triple makes up a valid cell
- and which weight should be assigned to the created cell

to an external `SegmentTripleFilter` object, which is accepted as a *template* argument to the `SegmentTripleCreator`. The latter can also decide, which fit strategy is employed for the final fit of the segment triples (see Section 6.4.5 below).

`TrackCreator` reduces the segment triple paths from the cellular automaton to a sequences of `CDCRecoHit3D` and transfers the first and the last fits from the path to the track.

`SingleSegmentTrackCreator` searches for leftover axial segments in the innermost superlayer to create tracks from them. As a preliminary approach we accept all segments with more than eight hits as probable tracks.

`TrackOrientator` takes care of the decision, if the tracks are to be interpreted as reversed. We always fix the orientation of the tracks as moving away from the interaction point.

`GFTrackCandCreator` translates the hit data from `CDCTracks` to `GFTrackCand` for Genfit and sets the initial values of the track fit.

Workers

To group the execution of the creators and the generic algorithms, we model the two major stages in the tracking by two worker objects,

- `FacetSegmentWorker` and
- `SegmentTripleTrackingWorker`,

which are in charge of the segment building and the track building respectively. The two worker instances provide the memory for the data structure objects to the creators. Hence the memory can be pooled and is not reallocated from one event to the next.

Module

Finally we implemented the tracking module, which handles the communication with the Belle II software framework. The `CDCLocalTrackingModule` is responsible for retrieval and sending the relevant data from and to the data store and invokes the two workers to carry out the actual tracking.

6.4.5 Creator filters and neighborhood choosers

We sensibly factored out the four major decisions in the building of the graph for the cellular automation namely

- which facets shall be formed
- which neighbors the facet shall acquire
- which segment triples shall be created and
- which neighbors of the segment triples shall be considered.

We can now optimize them in separate objects obeying specific interfaces.

Filters to the creators

The `FacetFilter` interface is used by the `FacetCreator` to determine, whether a combination of three neighboring wire hits is considered a valid facet. The definition of the interface is given in Listing 6.3.

The only method `isGoodFacet` returns a floating–point weight signaling, whether this specific cell is accepted and stored as a vertex. In case the cell is not accepted the return value should be the special constant `NOT_A_CELL` (which is a silent not-a-number in this implementation). In all other cases the cell is added to the pool of valid facets with its weight set to the returned value.

We also provide a concrete implementation with the `SimpleFacetFilter` class, which performs satisfactory for this step. It does the judgment of the facet by comparing the travel directions of the three tangent lines associated with the facet and returns the default weight of 3 for accepted facets.

```
class FacetFilter {

    /// Default constructor
    MCFacetFilter();

    /// Empty destructor
    ~MCFacetFilter();

    /// One method to decide, if certain facet shall be added as a valid
        cell.
    /// It returns a floating point CellWeight, in case the facet is
        accepted,
    /// or the special constant NOT_A_CELL, if the facet is not accepted.
    CellWeight isGoodFacet(const CDCRecoFacet & facet) const;

    /// Clears all remembered information from the last event.
    void clear();

};
```

Listing 6.3: The `FacetFilter` interface.

`SegmentTripleFilter` interface is employed by the `SegmentTripleCreator` to calculate the quality of triples of segments. Before the triple is check in conjuncture with the possible middle stereo segments, the compatibility of the pair of surrounding axial segments is evaluated. Therefore the interface of the `SegmentTripleFilter`, printed in Listing 6.4, has two methods according to these tasks.

The method `isGoodPair` indicates if the axial to axial pair shall be pursued further by a boolean decision. The second method `isGoodTriple` behaves similar to the method of `FacetFilter`, because it shall return a finite weight for accepted segment triples or `NOT_A_CELL` in order to prevent the addition of this cell. This method also has to set the fits of the segment triple in order to have them available in the neighborhood building phase.

The concrete implementation `SimpleSegmentTripleFilter` assesses the pairs and triples incorporating the Riemann and sz-fits. Despite of the mandatory check for the correct alignment of the segments, cuts on a variety of variables including

- distances of the segments to the circle trajectory of the axial segments,
- the angular deviation of the travel direction,
- the difference in transverse momentum of the start axial segment to end axial segment,
- the linearity of the sz-fit,

are possible and it is not a priori clear, which is best. Instead of testing all combinations by brute force, we seek to do a proper evaluation by comparing the variables and the

```
class SegmentTripleFilter {

public:

  /// Default constructor
  SegmentTripleFilter();

  /// Empty destructor
  ~SegmentTripleFilter();

public:

  /// First method to check, if a pair of axial segments is a good
      combination.
  /// Returns true of accepted pairs, false for unaccepted ones.
  bool isGoodPair(const CDCAxialRecoSegment2D & startSegment,
                  const CDCAxialRecoSegment2D & endSegment);

  /// Second method to decide, if certain a segment triple shall be
      added as a valid cell.
  /// It returns a floating point CellWeight, in case the triple is
      accepted,
  /// or the special constant NOT_A_CELL, if the triple is not
      accepted.
  CellWeight isGoodTriple(const CDCSegmentTriple & triple);

  /// Clears all remembered information from the last event.
  void clear();

};
```

Listing 6.4: The `SegmentTripleFilter` interface.

correct decision from Monte Carlo information directly, in order to find the best cut or multivariate combination. For a valid cell it returns the default weight, which is the total number of hits in the three segments.

Chooser to the building of the neighborhoods

The `NeighborChooser` interface for the `NeighborhoodBuilder` is shown in Listing 6.5, where `Item` has to be replaced by the class, for which we want to build a neighborhood, for instance `CDCRecoFacet` or `CDCSegmentTriple`.

The already mentioned three detailed questions,

1. where the lower bound of the block of possible neighbors is located in the sorted sequence,
2. when the current block of possible neighbor is exhausted and
3. whether the possible neighbor is acceptable as well as which weight shall be assigned to the edge,

correspond to the three methods of the interface. Because the first two methods,

1. `getLowestPossibleNeighbor` and
2. `isStillPossibleNeighbor`,

only reflect the general structure of the possible cell neighborhood, they will not be altered in the optimization process and we implemented them for `CDCRecoFacet` and `CDCSegmentTriple` in

- `BaseFacetNeighborChooser` and
- `BaseSegmentTripleNeighborChooser`,

from which the optimized choosers may inherit.

The latter only need to override the method `isGoodNeighbor`, which returns the weight of the graph edge to be made between the current item and its possible neighbor or `NOT_A_NEIGHBOR`, if no connection shall be made.

`SimpleFacetNeighborChooser` inherits from `BaseFacetNeighborChooser` and implements a concrete version of `isGoodNeighbor`, which judges the validity of the edge by comparing the angular deviation of the travel direction of the facet to its possible neighbor. The simple realization already proves to be acceptable, but can be further refined, since it only returns the default weight -2 for each connection with sufficiently small in angular deviation.

```
class NeighborChooser{

public:

  NeighborChooser(){;}   ///< Empty constructor
  ~NeighborChooser(){;}  ///< Empty destructor

  /// Getter for the lowest possible neighbor of the item according the
      sorting scheme of the items.
  const Item
  getLowestPossibleNeighbor(
      const Item & item
  ) const;

  /// Returns true, if the given item is still in the possible neighbor
      block.
  /// The lowest possible neighbor is also given as an argument,
  /// because it may prevent some computations from being redone.
  bool
  isStillPossibleNeighbor(
      const Item & item,
      const Item & neighbor,
      const Item & lowestPossibleNeighbor
  ) const;

  /// Returns the weight of the item neighbor connection.
  /// Signals not a number if no connection shall be made.
  /// The lowest possible neighbor is also given as an argument,
  /// because it may prevent some computations from being redone.
  CellWeight
  isGoodNeighbor(
      const Item & item,
      const Item & neighbor,
      const Item & lowestPossibleNeighbor
  ) const;

  /// Clear remembered information.
  void clear() const;

}; // end class
```

Listing 6.5: The `NeighborChooser` interface.

`SimpleSegmentTripleChooser` is a subclass of `BaseSegmentTripleNeighborChooser` implementing a concrete version of `isGoodNeighbor`, which judges the connection from one segment triple to a possible forward neighbor by comparing their z–coordinates and the z–slope of the sz-fit in the two triples. This particular decision is the least investigated, since we have concentrated on the prior ones. For valid neighbors, it returns the default weight, which is the number of hits in the overlapping axial segment between the current triple and its neighbor.

Chapter 7
Evaluation

> We can only see a short distance ahead, but we can see plenty there that needs to be done.
>
> Alan Turing

7.1 Organisation of the Monte Carlo information

With the local tracking approach we are modeling tracks detailed. Since every adjustment in a lower level stage can alter the optimal parameters at a following stage, it will be not sufficient to have only a single final value indicating the overall tracking efficiency. Besides the latter we need the ability to monitor the decisions in each step independently. Especially the filter and neighbor chooser objects shall solve the distinct problem of singling out the correct cells and edge connections in the graph. So instead of testing every combination of adjustable parameters for the overall tracking quality, we compare the available variables of the potential cells and neighbors to the Monte Carlo truth from the simulation, in order to fix the best quality cut or multivariate combination. Of course this implies, that the simulation correctly reflects each aspected we want to exploit for this distinction and care has be taken, if we suspect that this is not the case.

Indeed some flaws in the simulation of the central drift chamber have been found and have been consequently corrected by the responsable collaboration members, but no guaranty can be given that all errors have been detected yet.

Nevertheless we organised the Monte Carlo information obtainable through the data store form the used simulation into convinient lookup tables Having this facility available we can construct the filters and neighbor choosers:

- `MCFacetFilter`,
- `MCFacetNeighborChooser`,
- `MCSegmentTripleFilter` and
- `MCSegmentTripleNeighborChooser`,

based on the Monte Carlo information complementary to the `Simple-` implementations already mentioned. From these objects we can obtain the correct decision, whether to construct a corresponding call or graph edge. Subsequently we can use them for comparision of the Monte Carlo truth and the simple distinctions or in order to study the performance of a step of the algorithm, if the others were working perfectly.

7.2 Optimization of the facet creation

Having the Monte Carlo information available we can now begin to figure out the best distinction of valid cells and neighbors from false combinations, starting with the most fundamental `FacetFilter`, because the following decisions might be influenced by the concret choice at this point. A triple of wire hits with right–left passage information has three tangents and we can use the three angles between them to check for the correct alignment of the wire hits along the associated travel direction. For the correct alignment all three angles must be small. Hence we use the maximum of all three angle $\gamma_1, \gamma_2, \gamma_3$ as the variable to discriminate correct facets according to

$$\max\{\gamma_1, \gamma_2, \gamma_3\} < \text{cutoff value} \tag{7.1}$$

Stated differently we require all three angles to be smaller than the cutoff value. The Figure 7.1 shows the distribution of the maximum of the three angles for the correct facets stacked over the wrong combinations.

The distribution of the correct facets culminates at low values as expected. The background of incorrect facets is rather flat, but also increases towards lower values. The cut line, we assume best to reduce the background of false facets, while maintaining a high efficiency, is located at $5\pi/180$ rad, which includes the five left most bars of the diagramm. The following purity and efficiency can be achieved

$$\text{purity} = 49.76\% \tag{7.2}$$
$$\text{efficiency} = 91.97\% \tag{7.3}$$

with the cut and it got subsequently adjusted in the `SimpleFacetFilter`.

The other filtering and choosing steps have to be left for further investigation, because it was not possible to accomplish their optimization within the scope of this thesis. The premliminary implementations of the other stages are therefore unchecked and need to be reviewed in a similar way to the `FacetFilter`.

7.3 Tracking efficiency

Despite of striving to evaluate each step of the tracking we also seek to figure the overall tracking efficiency. This evaluation resides in a seperate module developed by a

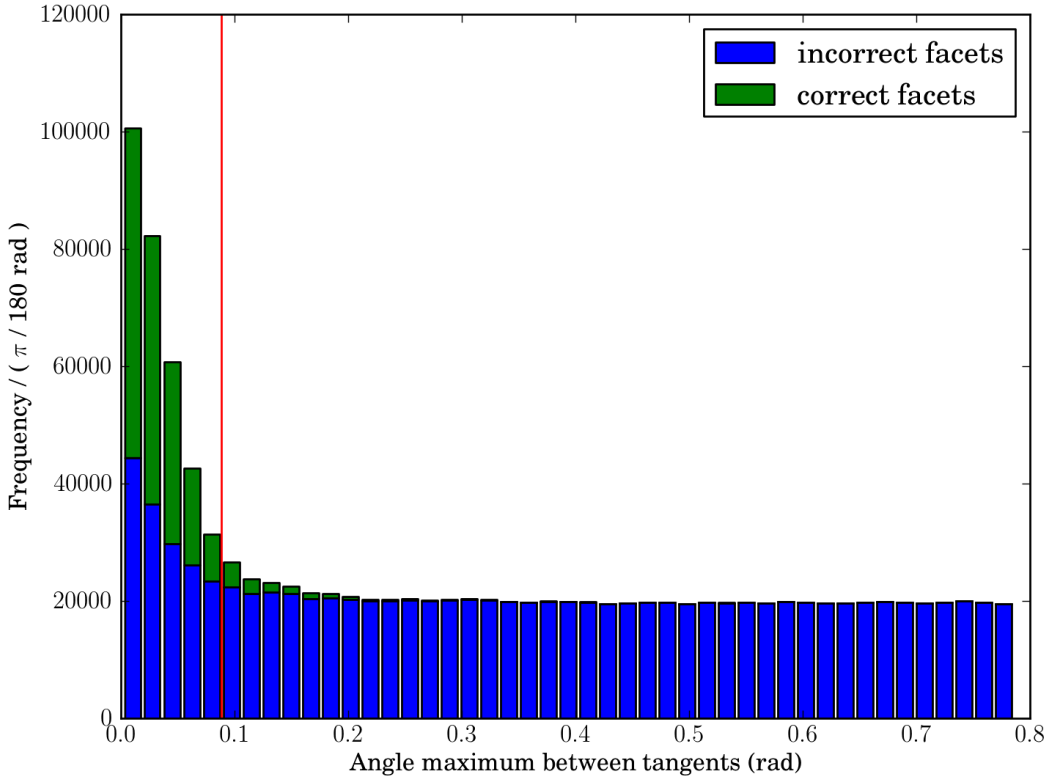


Figure 7.1: Histogram of the maximal value of the three angles between the three tangents to a triple of wire hits. The bin width is choosen to be $1° \equiv \pi/180$ rad. Correct combinations are shown in green, false ones in blue. The cut at $5\pi/180$ rad, marked red, seeks to maximize the purity without a significant loss in efficiency.

Run mode	FacetFilter and FacetNeighborChooser	SegmentTripleFilter and SegmentTripleNeighborChooser
1	MC	MC
2	simple	MC
3	MC	simple
4	simple	simple

Table 7.1: Tested combinations of filters and choosers in the two stages. *MC* indicates that the corresponding step has been supported with Monte-Carlo information in both the filter and the chooser, while the `Simple-` implementations using only information available from the actual detector.

colleague, Bastian Kronenbitter, since it forms the common base for the evaluation for the global and local tracking approaches, which are investigated at the institute. For this procedure simulated decays of the $\Upsilon(4S)$ resonance into two charged B mesons are utilized, where one of the B meson is further constrained to decay into two charged pions and a kaon involving an intermediate D meson, while the other is decaying generically. An event is counted as reconstructed, if the two pions and the kaons from one B meson are found by the tested tracking algorithm.

We ran the tracking in four different modes over the same sample of 1000 of the described events to derive a preliminary reconstruction quality of the current state of the development of the algorithm. Each of the runs has a different combination of filters and choosers for the graph creation, which can be found in Table 7.1. The first run mode carries out the combinatorics of the algorithm, but relies on the Monte Carlo information regarding the graph construction, while the following are increasingly realistic.

The number of the events counted as reconstructed in the four run modes are listed in Table 7.2. Additionally the table states the maximal achievable efficiency, which is obtained by constructing the tracks directly from the Monte Carlo hits. This maximal value is not 1000, because only particles leaving at least five hits in the central drift chamber are considered reconstructable. Hence if one of the relevant pions or the relevant kaon leaves the detector in the forward or backward direction, is stopped in the inner detectors or decays before reaching the central drift chamber, the event is not considered as reconstructable by the central drift chamber alone.

While the first stage already has satisfactory efficiency, this check reveals that further work has to concentrate on the optimization of the the second stage filter and neighbor chooser. Of course this is not suprising, since the parameters of the filter and chooser have not been crosschecked for their actual quality yet.

Run mode	Number of reconstructed events
Monte Carlo	707/1000
1	681/1000
2	656/1000
3	376/1000
4	294/1000

Table 7.2: Efficiency of the four run modes. The numbers are the count of the reconstructed events as is indicated by the evaluation module. We also quote the efficiency of the full Monte Carlo track finding as a maximal achievable reference.

Run mode	Average reconstruction time in [ms]
1	60
2	63
3	113
4	116

Table 7.3: Time consumed by the tracking algorithm per event in the four different run mode. The times include the building time for the Monte Carlo information

7.4 Performance

We also tested the time performance of the implementation to get an impressions of the time spend for the tracking. Measured with the basic profiling mechanism of the Belle II analysis software framework, the four checked run modes need the average times per event as listed in Table 7.3. Again these numbers have to be taken with a grain of salt, because the second stage is only preliminary set up. Also the time spend to build the lookup tables from the Monte Carlo information has to be subtracted, since this will not be necessary in the actual run. We analysed the time consumed in run mode 2 with the *Valgrind* profiling tool, which showed that 60% of the time in this run can be attributed to the building of the lookup tables. The performance of the algorithm is therefore reasonable for the current development status.

7.5 Conclusion

Within this thesis we developed the scafford of a tracking algorithm fully based on the local tracking principles. The development of the outlined building blocks like tra-

jectories, cellular automation and the various hit entities has been done, in order to form a solid base for further enhancements in any direction, we might consider worth investigating. The performance and efficiency are not yet optimal, but in an expectable range considering the status of the development and well in reach of desirable values. Especially the adjustments for the building of cells and neighbors are close to be completed in the continued effort to provide a complete tracking algorithm to the central drift chamber of the future Belle II experiment.

Bibliography

[1] C. Höppner et al. "A novel generic framework for track fitting in complex detector systems". In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 620.2-3 (2010), pp. 518 –525. URL: http://www.sciencedirect.com/science/article/pii/S0168900210007473.

[2] Belle II Collaboration. *Belle II Technical Design Report*. Version 1. Nov. 1, 2010. arXiv:physics/1011.0352v1 [physics.ins-det].

[3] A. Abashian et al. "The Belle detector". In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 479.1 (2002), pp. 117 –232. URL: http://www.sciencedirect.com/science/article/pii/S0168900201020137.

[4] KEK High energy accelerator research organisation. *New electronics tested for Belle II central drift chamber*. Mar. 23, 2010. URL: http://legacy.kek.jp/intra-e/feature/2010/BelleIICDCDesign.html.

[5] David J. Lange. "The EvtGen particle decay simulation package". In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 462.1-2 (2001), pp. 152 –155. URL: http://www.sciencedirect.com/science/article/pii/S0168900201000894.

[6] S. Agostinelli et al. "Geant4 - a simulation toolkit". In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 506.3 (2003), pp. 250 –303. URL: http://www.sciencedirect.com/science/article/pii/S0168900203013688.

[7] N.I. Chernov and G.A. Ososkov. "Effective algorithms for circle fitting". In: *Computer Physics Communications* 33.4 (1984), pp. 329 –333. URL: http://www.sciencedirect.com/science/article/pii/0010465584901371.

[8] J.F. Crawford. "A non-iterative method for fitting circular arcs to measured points". In: *Nuclear Instruments and Methods in Physics Research* 211.1 (1983), pp. 223 –225. URL: http://www.sciencedirect.com/science/article/pii/0167508783905756.

[9] M. Hansroul, H. Jeremie, and D. Savard. "Fast circle fit with the conformal mapping method". In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 270.2-3 (1988), pp. 498 –501. URL: http://www.sciencedirect.com/science/article/pii/016890028890722X.

[10] Veikko Karimäki. “Fast code to fit circular arcs”. In: *Computer Physics Communications* 69.1 (1992), pp. 133 –141. URL: http://www.sciencedirect.com/science/article/pii/001046559290134K.

[11] Björn Lillekjendlie. “Circular Arcs Fitted on a Riemann Sphere”. In: *Computer Vision and Image Understanding* 67.3 (1997), pp. 311 –317. URL: http://www.sciencedirect.com/science/article/pii/S1077314297905294.

[12] A. Strandlie, J. Wroldsen, and R. Frühwirth. “Treatment of multiple scattering with the generalized Riemann sphere track fit”. In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 488.1-2 (2002), pp. 332 –341. URL: http://www.sciencedirect.com/science/article/pii/S0168900202004655.

[13] V. Blobel and E. Lohrmann. *Statistische und numerische Methoden der Datenanalyse*. Teubner-Studienbücher : Physik. Teubner, 1998. ISBN: 978-3-935702-66-9. URL: http://www.desy.de/~blobel/ebuch.html.

[14] J.H. Wilkinson. *Rounding Errors in Algebraic Processes*. Dover books on advanced mathematics. Dover Publications, 1963. ISBN: 9780486679990.

[15] Gaël Guennebaud, Benoît Jacob, et al. *Eigen v3*. http://eigen.tuxfamily.org. 2010.

[16] Oksana Lutz. “Search for $B \to h^{(*)}\nu\nu$ at Belle and development of track finding for Belle II”. PhD thesis. Fakultät für Physik des Karlsruher Institut für Technologie (KIT), 2012.

[17] A. Strandlie and R. Frühwirth. “Error analysis of the track fit on the Riemann sphere”. In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 480.2-3 (2002), pp. 734 –740. URL: http://www.sciencedirect.com/science/article/pii/S0168900201012281.

[18] I. Abt et al. “CATS: a cellular automaton for tracking in silicon for the HERA-B vertex detector”. In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 489.1-3 (2002), pp. 389 –405. URL: http://www.sciencedirect.com/science/article/pii/S0168900202007908.

[19] I. Kisel et al. “Cellular automaton and elastic net for event reconstruction in the NEMO-2 experiment”. In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 387.3 (1997), pp. 433 –442. URL: http://www.sciencedirect.com/science/article/pii/S0168900297000971.

[20] Belle II Collaboration. *Introduction to the Belle II analysis framework*. Feb. 27, 2013. URL: https://belle2.cc.kek.jp/~twiki/bin/view/Computing/Basf2Introduction.

Danksagung

Schlussendlich zum Abschluss der Jahre des Studiums der Physik möchte ich mich bei allen bedanken, die zur Vollendung dieser Arbeit beigetragen haben.

Zunächst möchte ich mich bei Prof. Dr. Michael Feindt für die Betreuung dieser Diplomarbeit und seine Förderung darüber hinaus bedanken.

Danke gebührt auch Prof. Dr. Ulrich Husemann für die Übernahme des Korreferats.

Herzlichen Dank an Dr. Martin Heck und Dr. Thomas Kuhr für die fundierte und aktive Unterstützung und Beratung während dieses Jahres.

Für die entspannte und unterhaltsame Atmosphäre, ins insbesondere im Zimmer 9-2, bedanke ich mich bei der B-Physikgruppe des Institutes für experimentelle Kernphysik. Insbesondere bedanke ich mich bei Bastian Kronenbitter, der stets ein offenes Ohr für jegliche Fragen hatte, sowie bei Christian Pulvermacher, Oksana Lutz und Manuel Kambeitz für die Korrektur der Kapitel.

Darüber hinaus möchte ich mich besonders bei meinen Freunden und liebgewonnen Menschen hier und in aller Welt bedanken, die meine Studienzeit auf viele Weise bereichert haben.

Ebenso möchte ich meinen Eltern und meiner ganzen Familie danken, von der ich auch in schwerer Zeit stets unterstützt würde und auf die ich immer verlassen konnte.