Design and Implementation of Alignment software for a Digital Tracking Calorimeter used for proton CT

Rune Almåsbakk

Master's thesis in Software Engineering at

Department of Computer science, Electrical engineering and Mathematical sciences,
Western Norway University of Applied Sciences

Department of Informatics,
University of Bergen

June 2022

Western Norway
University of
Applied Sciences

Abstract

Computed Tomography, also called CT, is a diagnostic method used to create images of a patient's body. These images can be used to identify or discover illness in the imaged patient. Tomography is nothing new and has been around since the early 1920s [1]. Today such images are also used to plan and execute radiation therapy.

A prototype of a Proton Computed Tomography(pCT) system is currently under development by a research team at the University of Bergen and Western Norway University of Applied Sciences. This project introduces proton-based imaging instead of the standard photon-based sensors in today's CT machines. Protons are used in-place-of photons based on the underlying properties of a proton particle and the reduction in overall time one round of treatment would take for each patient.

The Proton Computed Tomography system is built using a chip called ALice PIxel DEtector(ALPIDE) developed at CERN. Several ALPIDE chips are used to construct the sensor, which can introduce accuracy issues in the output data, as these chips can only be mounted with limited mechanical precision. This thesis introduces the software theory behind a system used to align the chips based on output data from the sensor.

Acknowledgements

First and foremost, I would like to thank my supervisors, Professor Håvard Helstrup and Professor Johan Alme. I would also like to include Matthias Richter, you have all been of great help and a pleasure to collaborate with. You have been consistently quick at answering all my questions and continuously given feedback on my progress, for which I am immensely grateful.

I would also like to thank Viljar Eikeland and Alf Kristoffer Herland for giving me an introduction to the details of the pCT project. Also, a big thanks to my family and friends for being a constant support through this process.

Rune Almåsbakk
Bergen, June 1, 2022

Contents

List of Figures

List of Tables

Listings

Glossary

Bragg peak The peak energy deposition of a particle traveling through matter, right before it comes to a rest.

calorimeter A device used to measure the quantity of heat transferred to or from an object.

Docker A platform as a service product that runs software in "self-sustaining" packages called containers. Allowing anyone to run software packages without having to install any dependencies[2].

layer This is the largest element of the DTC constructed from 12 staves. Forty-three of these combine into the full DTC.

offset Referring to elements inside the DTC being shifted or misaligned in relation to every other element.

phantom Object used as an obstruction for particles during testing and simulation of the DTC. Imitating human tissue.

readout unit Component responsible for gathering and parsing incoming ALPIDE data. One unit for each layer.

residual The difference between an original track and its predicted track. Measured in the distance from the original hit to the linear regression line of that track.

stave A DTC element constructed from 9 ALPIDE chips. 12 of these combines into a layer.

Valgrind A framework used to create tools able to detect bugs and profile memory management and function calls[3].

Acronyms

ALICE A Large Ion Collider Experiment.

ALPIDE ALICE pixel detector.

CERN derived from the name Conseil européen pour la recherche nucléaire meaning The European Organization for Nuclear Research.

CI/CD Continuous Integration/Continuous Delivery: A automation process for developers to ensure new code is built and tested properly, meant to fix issues with integrating new code in a larger codebase.

CT Computed Tomography.

DTC Digital Tracking Calorimeter.

ITS Inner Tracking System.

LET In dosimetry, linear energy transfer (LET) is the amount of energy that an ionizing particle transfers to the material traversed per unit distance.

LHC Large Hadron Collider.

MC Monte Carlo. Used here to refer to Monte Carlo simulation.

OLS Ordinary Least Squares. Type of linear regression model.

OS Operating System: A interface between a computer user and computer hardware.

pCT Proton Computed Tomography.

TC Transition card. Part of the readout electronics in the Bergen pCT system.

WSL Windows Subsystem for Linux: A Windows system allowing emulation of a Linux system inside of Windows.

Chapter 1

Introduction

This section will introduce the information needed to understand the problem. The section will have little to no connection with software engineering. However, it will be crucial to understand the more extensive project this thesis is built upon and create an understanding of why the research has to be done.

1.1 Background

The Bergen pCT project explores a new method to perform computed tomography(CT), mainly the possibility of a fully proton-based CT scanner. This differs from today's CT scanner, where photons are the source used to create images. There are several benefits of using protons as the source, including reducing errors related to current imaging approaches for particle therapy treatment planning. This thesis will focus on the detector used to collect data from proton particles. The detector is constructed by using several ALPIDE chips from the ALICE(A Large Ion Collider Experiment) experiment at CERN. The goal is to be able to do CT scans of patients using this new detector.

The work done on this thesis is part of the Bergen pCT collaborations effort to reduce errors related to data-gathering from the detector mentioned earlier. This work will lay the foundation necessary for further development on the topic.

1.1.1 Radiation therapy issues

There are many factors in today's radiation therapy that can introduce errors. These errors may cause damage or unwanted side effects to a patient. The factors include conversion errors when going from photon-stopping-power to proton-stopping-power and the possibility for organs to move between the imaging process to the proton therapy process [4].

The fact that the target(e.g., cancerous cells) can move inside the patient between the imaging step and the point where therapy begins opens the possibility of radiation being delivered to otherwise healthy tissue, increasing the risk of new development of cancerous cells.

Taking CT scans also introduces radiation to healthy tissue because of the properties of photons, which is also a reason why protons are researched as a replacement for photons during CT scans, discussed further in section 1.1.2.

1.1.2 Proton Computed Tomography

When dosage plans are made by the CT machine, the patient is under constant effect of radiation. The most optimal procedure would include no radiation, but this is not possible. Therefore the minimal amount of radiation possible is the goal. Figure 1.1 shows a comparison between photons and protons as source used for imaging, which indicates that protons introduce less radiation than photons to healthy tissue.

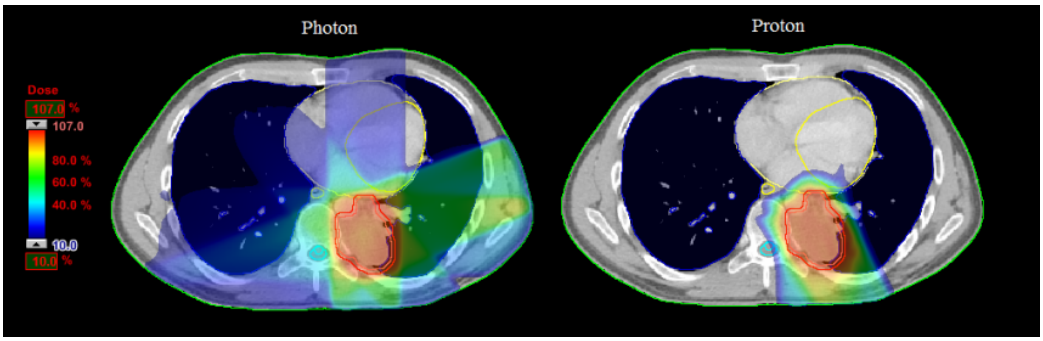


Figure 1.1: *A comparison between two dose plans for irradiation of a paravertebral sarcoma in the lung, overlaid on CT images. Note the difference in volume between the low dose regions (the so-called low dose bath) visualized as blue areas, substantially smaller in the proton plan [5].*

In Figure 1.2 you can see the properties of protons and photons clearer. Less dose is delivered to the patient while the proton pass through tissue, but a higher dose is delivered where the proton stops. This energy deposit pattern is described by the "Bragg Peak", known from the physics of ionizing particles. Photons, on the other hand, deliver a dose to the tissue that is always highest at the entry point, and diminished through the length of the photon path. The fact that protons stop at a given depth (opposite to photons having an exponential decline), clearly shows the advantage of using protons for imaging and treatment.

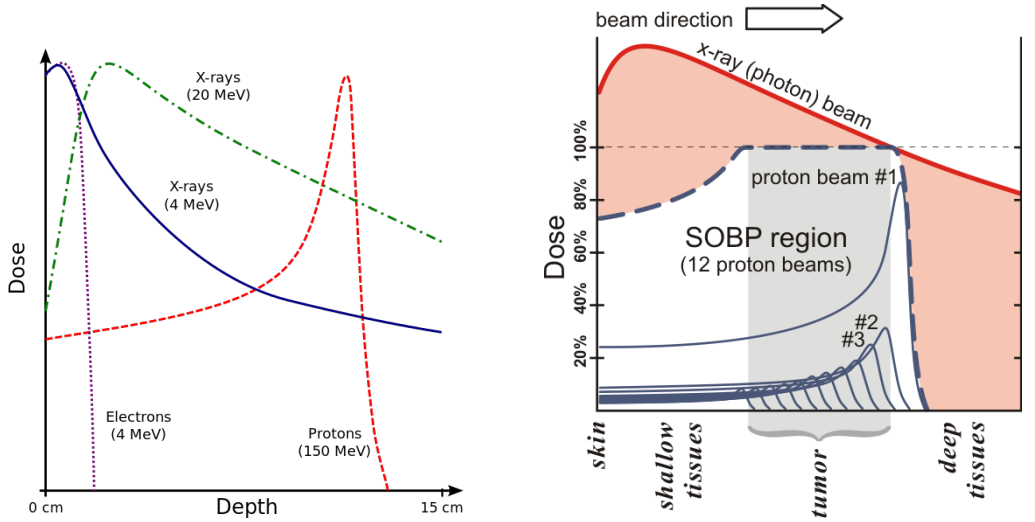


Figure 1.2: *(a) Averaged LET depth dose deposition for typical forms of ionizing radiation, assumed to impact in statistically relevant number from free space into a solid-state target material resembling tissue (IRCU44) at a given impact energy [6]. (b) showing a thematic diagram showing dose as a function of depth for overlay of proton radiotherapy and x-ray radiotherapy to facilitate a comparison of the two radiotherapy methods [7].*

The Bragg peak will happen inside the detector during a CT scan, lowering the total dose applied to the patient. Protons deliver less radiation during travel than photons, indicating that most of the dosage will be absorbed by the detector and not the patient. When doing a scan, the energy of the protons will be higher than what is used doing therapy, ensuring that protons

will not stop inside the patient. In the case of therapy, from Figure 1.2(b), one can see that most of the radiation from the proton beam is delivered inside the tumor, dropping off drastically right after. The fact that protons can reduce radiation this much is one of the main factors for why the Bergen pCT project is being developed. By reducing the total radiation delivered to a patient during a scan, and reducing the inaccuracy introduced by conversion between photon scans and proton treatment, proton CT can reduce the risks involved in position measurement and proton therapy[5].

The scope of the Bergen pCT project is to develop a Digital Tracking Calorimeter(DTC) capable of collecting data from a proton source, with a goal of using this in a therapy machine. Its main concepts are visualized in Figure 1.3. This machine will be installed at the new proton center at Haukeland hospital[8].

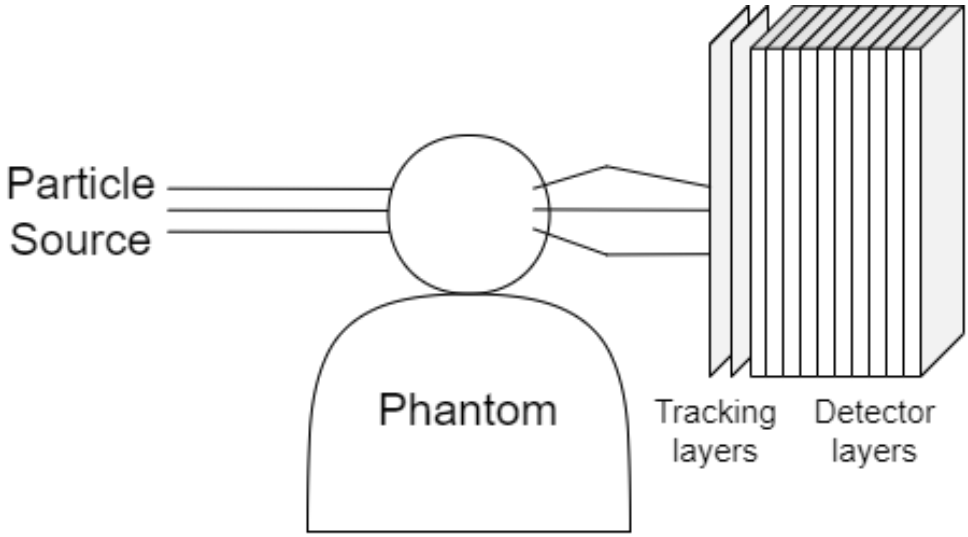


Figure 1.3: *A proton CT setup with a Digital Tracking Calorimeter and proton source [9].*

1.1.3 The Alice Pixel Detector

The ALICE detector in CERN Switzerland is a detector that is used to research heavy-ion physics. It is part of the Large Hadron Collider (LHC) and is used to study particles at very high energy densities[10]. In 2019-2020 a planned upgrade of the LHC was done where a new ALPIDE chip was

introduced. This chip was developed to withstand the harsh environments inside the ITS(Inner Tracking System)[11].

The ALPIDE chip is a pixel-based particle sensor for ionizing particles. It is constructed like a grid with height 512px and width 1024px where each pixel is 28 µm x 28 µm. The chip can read particle hits in a binary hit/no-hit fashion and was originally designed for the ALICE experiment at CERN[12]. It is now used in the Bergen pCT project because of its high accuracy and low power consumption, together with its radiation resistance which makes it able to last for a considerable amount of time before needing to be replaced[11].

1.2 Problem Description

As the ALPIDE chips are mounted to create a Digital Tracking Calorimeter there is no way to ensure a perfect mechanical alignment of the chips. A software module capable of post-construction realignment is therefore necessary to limit the data errors resulting from construction and the general life-cycle of the Digital Tracking Calorimeter. This paragraph will describe the DTC, and present the alignment problem.

1.2.1 pCT Digital Tracking Calorimeter

The Bergen pCT collaboration was established at the University of Bergen among many institutions across the world to design and build a prototype pCT scanner. For this prototype, a sensor referred to as a Digital Tracking calorimeter is under development. The DTC should be able to do tracking and residual energy measurements to make scanning simpler[13].

The DTC is a layer-by-layer structure where each layer is comprised of several smaller components. The smallest component being the previously described ALPIDE chip. The detector has an aperture of 27 cm width by 16.6 cm height with a total of 43 of these 27x16.6 layers stacked. A layer consists of 12 staves, and a stave consisting of 9 ALPIDEs side by side. The active area of the sensor corresponds to a width of 9 ALPIDEs, and a height of 12 ALPIDEs, constructing a grid. The total amount of ALPIDEs in the DTC is 43x12x9 or 43 layers, 12 staves per layer and 9 chips per stave, resulting in 4644 ALPIDEs.

Figure 1.4 shows the general structure of the Bergen pCT system. Each

of the 43 layers has one dedicated transition card(TC) being an interface between the ALPIDEs and the rest of the system. Every odd layer is rotated to the opposite side to make room for the readout electronics. From the construction of the DTC each pixels relative position is related to what layer and stave it is located. Fig 1.4, starting bottom-up, each odd stave(black) is rotated or "flipped" to the opposite side of the layer, altering the position of each pixel relative to the stave above and below(brown).

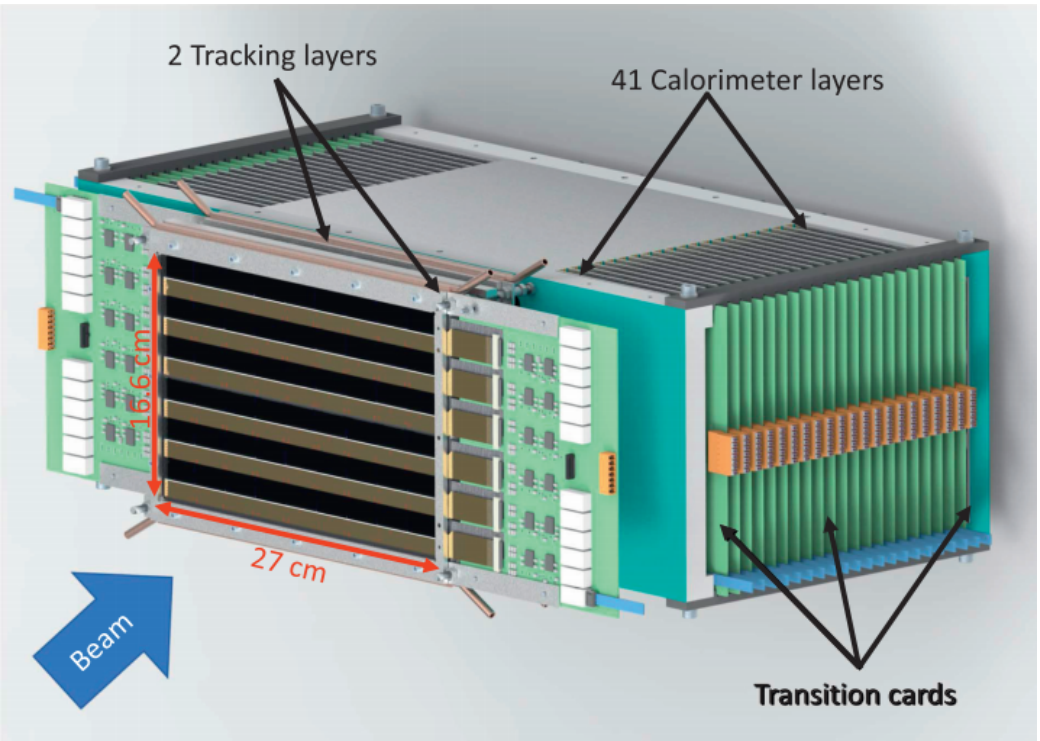


Figure 1.4: *The general structure of the Bergen pCT system [13].*

1.2.2 Alignment

During assembly of the DTC described in section 1.2.1, the positional precision of chips mounted to staves, staves mounted to layers, and layers finally constructed into the detector can not be done with perfect accuracy. The concept of an alignment algorithm is therefore investigated in this thesis. The idea for the alignment algorithm is a software module capable of producing correctional alignment parameters for individual elements inside the

detector(e.g. chip, stave or layer).

Even after construction and alignment have been done, several misalignments can occur over time. The detector can be affected by, e.g., vibrations or heat fluctuation, which will produce offsets within the detector during its lifetime. Some parts will be more affected than others because of how the elements are mounted. In general, alignment has to be done fairly often to ensure accurate data from the detector.

In a general case, the detector would have some object between itself and the source of the particles, but this is not the case when alignment has to be done. If the particle is hindered by anything in its path, it would increase the noise in the data and decrease the algorithm's accuracy. The noise would interfere with the assumption of straight path particles through the detector and therefore needs to be minimal.

The concept of this alignment module will be discussed as the primary topic in this thesis, more precisely, the choices made during development and why they are optimal for usage in both alignment and the pCT project as a whole.

1.3 Research questions

How to create an alignment algorithm that can find structural misalignment in a pixel-based sensor given simulated data?

Investigate the possibility to produce offset parameters for the Bergen pCT Digital Tracking Calorimeter given simulation data. Implement a module that can produce these offsets efficiently and discuss any issues that arise with the chosen method.

What offset parameters are needed for each detector element, and what is an optimal way to store these parameters for further usage?

This thesis will also discuss the parameters needed to align each element in the detector and what an optimal way to store these parameters could be.

1.4 Outline

Chapter 2 - Software Background

An explanation of the theoretical background will be done in this chapter, more precisely of what design practices were chosen and used. Prototyping, simulation, and other practices will be the main focus.

Chapter 3 - Design and Implementation

In this chapter, the design of the system and the general implementation of the system will be discussed. The choices made before development began and an explanation of algorithms will be made here.

Chapter 4 - Analysis and assessment

This chapter will cover the analysis of the resulting time usage and accuracy of the alignment algorithm and the availability of the resulting offset parameters.

Chapter 5 - Conclusion

This chapter will summarize the results generated by the work and experiments done for this thesis.

Chapter 6 - Further Work

General suggestions for further development of the system will be listed here. It will include performance and accuracy improvements and several other possible improvement cases.

Chapter 2

Software Background

This section will give an introduction to general concepts and practices related to this thesis. This includes technologies and methods specific to the pCT project while also covering broader software design patterns. The presented knowledge is needed to follow along with the implementation description and experiments done during testing and analysis. The section will finish with an introduction to related work and used methodologies.

2.1 Concepts and Tools

2.1.1 Prototyping

Software prototyping is often used to get insight into the properties and functionality of an idea or concept to see if it works or performs as expected. The outcome is often an incomplete version of the software application, but one gains information to develop and implement new features. In the end, one should have sufficient information about the system and know what to do and not do for continued development[14].

The intended goal for this thesis is to create such a prototype to gain insight into what is needed for alignment. The implementation is the first draft of an alignment system, acting as a framework for further development. The framework will be researched further and improved alongside the improvement of the Bergen pCT project.

The target of this prototype is the alignment module of the pCT software package. To implement the prototype, a set of requirements and specifications was made before development began, as discussed further in section 3.4.1.

2.1.2 Simulation

The use of simulated data can be applied in many critical areas. Using simulated data, one can find issues with the system and fix them before they become more significant problems. The simulated data gives insight into the system's behavior and how the system's components work together. It can also bring further information about the process and architecture and indicate what needs to be fixed before more time or cost is invested[15].

While writing this thesis, the DTC is still under development. Therefore acquiring actual data for testing is not possible. Because of this, the alignment prototype relies exclusively on simulated data. For the case of alignment, the usage of simulated data is actually the preferred method. The prototype is based on the concept of misalignment represented in the output data from the readout unit. To analyze the prototype results, one needs to verify the result by knowing where the misalignment is located. One also needs to know by how much the element is misaligned. This misalignment can not be verified within a physical sensor. However, we can choose where the misalignment is located using simulated data. Simulation of a misaligned sensor is the reason verification of the alignment module is possible.

Verification is done by running a "blind" test on data that has hidden misalignment. If the alignment module finds the correct offset parameters, it indicates that the module works as expected.

2.1.3 Monte Carlo Simulation

Monte Carlo simulation is a method used to generate data based on random sampling. It is used to generate results from experiments where the expected results are not known in advance[16]. The Monte Carlo simulation method uses randomness to compute results, and it is often used in situations where there is no direct way to produce the same data. Like in the case of getting readout data from the pCT detector mentioned in section 2.1.2[17].

In the Monte Carlo simulation, a probability distribution of all interactions of particles is used to simulate how each particle moves and interacts within the volume of the simulation "world." From this, it is possible to log the history of each individual particle, including the path and the energy of that particle[18].

2.1.4 Track Reconstruction

The concept of a track is the path of a particle traversing through the DTC giving a single x and y coordinate from every layer. This concept is heavily used throughout this thesis and is essential for the entire alignment module. Tracks are visualized in Figure 3.5 for reference.

All data collected from the DTC has no order and is just data about one single hit for a particle hitting an ALPIDE chip. Because the hit information cannot be combined into tracks based on the timing of hits, a track reconstruction algorithm has to be applied to find the hits that are most likely to be part of the same track. The track reconstruction problem is connected to the alignment problem because the position of elements in the detector affects the distance between hits.

To create tomography images from the data, this scatter of single points in the three-dimensional detector needs to be combined into tracks with as little error as possible. The process of doing this will include applying the alignment offsets to the data before track reconstruction. One problem is that the track reconstruction algorithm relies on already aligned data. A incremental process of reconstruction $\rightarrow$ alignment $\rightarrow$ reconstruction may therefore be necessary. More information on track reconstruction can be found in section 2.3.

2.1.5 Linear Fit

Linear fit, also known as linear regression, explores the relationship between an observed variable and a predicted variable. It is often used to make predictions when the predicted value is closely related to some external factors. E.g., house pricing based on how many rooms are in that house. The process of doing this regression is to create a linear function that has the least-squared error based on a set of training points[19]. A simple formula can express the resulting estimator:

$$\mathbf{y} = \beta_0 + \beta_1 \mathrm{X},$$

Where $\mathbf{y}$ is $n \times 1$ vector of the response variable of the n observations, β_0 the intercept value, β_1 is a $p \times 1$ vector of unknown parameters and X is an $n \times p$ matrix of regressors and contains the observed explanatory variables. β_0, β_1 are the unknown values that is to be estimated.

2.1.6 ROOT: Data Analysis framework

The ROOT framework is widely used among physicists that need to process large data sets. It is a framework that can be used to read, save, and process data, while it can also be used as a visualization tool. The framework is built around a specific tree-structured file type with the extension ".ROOT" [20]. The Root file type is the chosen file type for storing output data for the pCT system alongside binary data.

This framework is used in the alignment module not as a storage tool but rather as an analysis tool with its histogramming options. Using a Gaussian fit over a histogram, the mean value of the fit can be calculated and used as the end product in the alignment algorithm. More on this in section 3.4.4.

2.2 Software design Patterns

As explained in the book Design Patterns: Elements of Reusable Object-Oriented Software (1994)[21]. The difference between experienced object-oriented designers and new designers is how they reuse techniques. An experienced designer will not solve problems from the ground up but rather reuse past experiences and patterns. These patterns are often proven to be reliable and make designs better. They are designed around specific problems and make the development process simpler.

2.2.1 Dependency Injection Pattern

In software engineering, a design pattern called dependency injection is often used. This pattern is used to deliver services where they are needed. An object that depends on another object to perform its task, receives this dependency through a mediator. The mediator is responsible for managing

every dependency injected into this object. This reduces coupling between the objects making the code cleaner and more readable[22].

By reducing the number of responsibilities of an object, the code becomes more reusable. If an object does not have to care about how or where its dependencies are created, it becomes more testable and maintainable. Furthermore, because every dependency is handled by a single component, less boilerplate code is created[23].

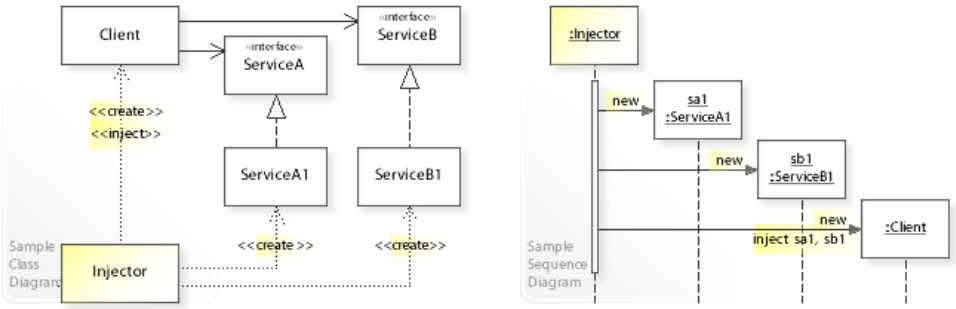


Figure 2.1: *Illustration of dependency injection[24].*

There are three types of dependency injection, Constructor injection, Setter injection, and Interface injection. For this thesis, the constructor injection method is used. This method requires the dependent object to have the dependency as a parameter in its class constructor. This ensures that the dependency is in a valid state upon creation of the dependent object[23].

2.2.2 Policy-based Design

Policy-based design is a design approach with what is referred to as policies as its central concept. It resembles the strategy-pattern, with its difference being a compile-time variant rather than runtime[25].

A policy class is an interface containing: inner type definitions, member functions, and member variables. This interface defines the functionality of the specified policy and makes the system design highly customizable. By using policies, the detailed functionality of the system can always be changed without any significant consequences. It also makes these changes

easy as it does not affect other parts of the system, ensuring safe and efficient development[26].

The concept of a policy-based design is centered around a host class acting as a behavioral manager. The host class is responsible for the final functionality of the system by using the policy classes. Each policy class is responsible for its behavior but is restricted by some rules set in the system's design. The system sets these rules and has no specifications set by the design pattern. In the case of the alignment module, these rules are based on the input and output each policy must follow. Apart from that, what the policy does has no clear definition. As long as the policies consume and produce the correct input/output, the system should work without issues related to the policy interface[25]. Figure 2.2 visualized the concept of policy-based design through the strategy-pattern.

This design pattern makes the system very modular. By keeping code decoupled, rewriting stays manageable as each policy can easily be replaced without replacing or changing the other policies. The design pattern also allows several different approaches to be utilized simultaneously if necessary. The usage of the pattern will be explained further in section 3.4.

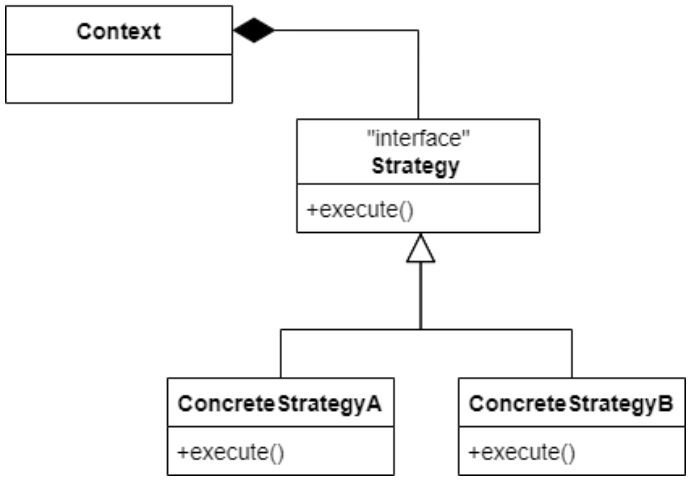


Figure 2.2: *General example of the strategy pattern[27].*

2.2.3 Generic Programming

Generic programming is a programming style where classes or functions are written in terms of unspecified types that are then provided as parameters upon the definition of that class or function. This allows writing functions that are different in the types they operate with but are equal in the way they are written, reducing code duplication. This concept is most commonly referred to as generics but in C++, it is known as *templates*[28]. This programming style is mentioned as *parameterized types* in Design Patterns: Elements of Reusable Object-Oriented Software[21].

Generic programming is defined in Musser & Stepanovs *Generic Programming**(1989)[29] as:

> *Generic programming centers around the idea of abstracting from concrete efficient algorithms to obtain generic algorithms that can be combined with different data representations to produce a wide variety of useful software.*

By generic programming, what is meant is the definition of algorithms and data structures at an abstract or "generic" level. A great example of this is presented as a List data structure in Listing 2.1 where the type stored in the list is specified by the typename T. Example given here is Animal and Car as types "replacing" T.

```
1 template<typename T>
2 class List {
3   // Class contents.
4 };
5
6 List<Animal> list_of_animals;
7 List<Car> list_of_cars;
```

Listing 2.1: *C++ template/generics example.*

One now has a List data structure to store objects with, no matter the type of the object. Because of this, one no longer needs to create the same list structure for every type in the system.

2.3 Related work

This thesis is built on top of work done over several years, from the design of the DTC to implementations of protocols. Some of the work related to this thesis is listed here.

A High-Granularity Digital Tracking Calorimeter Optimized for Proton CT [13]

This article by Johan Alme covers how the digital tracking calorimeter is constructed and how it functions with the rest of the system.

Scalable Readout for Proton CT[30]

This thesis focuses on much of the logic behind the readout process of the pCT system, specifically parsing of particle events recorded by the DTC for output file writing.

Proton Tracking Algorithm in a Pixel Based Range Telescope for Proton Computed Tomography[31]

&

Optimization of the Track Reconstruction Algorithm in a Pixel Based Range Telescope for Proton Computed Tomography[32]

These articles focus on track reconstruction algorithms for a proton-based particle detector and explain how this is done in more detail.

2.4 Methodology

This thesis is a tiny part of a larger project done by the pCT collaboration. Both quantitative and qualitative evaluations have been done to produce something usable for the project. This ensures consistency in formats across modules and systems in the project. Weekly group meetings and consistent discussion on several other channels made it easier to progress and discuss solutions to arising problems.

Pinpointing the exact research methodology is, in many cases, very difficult. Even though both qualitative and quantitative evaluations were done, the research was not limited to these methodologies. The most obvious comparison would be exploratory research, where the approach is more flexible

and investigative. The results are topic-related, which helps with the further development of this research.

Chapter 3

Design and Implementation

This section will explain the design and implementation of the alignment module in detail. This includes explanations of the more extensive pCT-Online software, test environments, and a detailed explanation of the implemented algorithm used for alignment. Several external tools will also be explained regarding how they are used for developing a prototype.

3.1 pCT-online

3.1.1 The pCT-Online package

The pCT-online package contains all modules related to online readout. As opposed to offline, this means data is fetched, parsed, and stored while the DTC is actively registering particle hits during a scan. The code contains several protocols, data structures, and algorithms related to the readout chain.

The alignment module is part of the pCT-Online software package as a tool used to create more accurate data. It is not expected to be used in the readout chain.

3.1.2 The Readout chain

The readout chain is a collection of several components that together makes the system responsible for everything related to data collection from the

DTC. The readout chain is visualized in Appendix C, giving a simple explanation of all involved parts.

Because of high radiation around the DTC, multiple TCs are used as an interface between the layers of the DTC, the readout units, and power control unit. The TC gives the possibility to increase the distance between sensible electronics and areas of high radiation, decreasing both cost and development time. The power control unit is responsible for managing the power delivery to the DTC but is not directly connected to the data readout.

The readout units are responsible for the initial data collection and parsing before the data is sent to a general control system. It needs to collect, parse, and deliver the data as quickly as possible to avoid losing critical information. When a particle hits an ALPIDE, the hit information is initially processed on the readout unit, which could easily overflow if the data is not parsed fast enough. When the central control unit receives the data, further heavier calculations can be done on it as it has a more permanent storage option[30].

3.1.3 The pCT-Online Continuous Integration pipeline

A pipeline has been implemented in the pct-online repository to ensure the software is in an executable state. This pipeline uses Docker to build and test code pushed to the "dev" branch. It is a common strategy used for CI/CD to ensure that any merges to a "main" branch are free of potentially software-breaking bugs before being pushed to production. The Docker service also ensures that the pct-online package can run on any computer, which confirms that any new implementations are not system-specific. The Docker building step of the pipeline also enables cross OS development, making it easier for anyone participating in the codebase to develop on any system. E.g., the Docker version runs on the intended CentOS distribution of Linux. However, most of the development done in this thesis is done on either the Ubuntu distribution or Windows Subsystem for Linux(WSL), which is part of the Windows operating system.

3.2 The Test Environment

A set of tools and simulation scripts was created to understand, verify, and create data. These tools were used as a test environment for the alignment

module to ensure things worked as expected.

3.2.1 The Toy simulation script

To start development on this prototype, a way to create usable data with the possibility of inputting offsets was necessary. It was not yet implemented in the Monte Carlo simulation from section 3.2.2, so a very simplified "toy" simulation script had to be implemented to generate data. This script does not include any accurate particle simulation but creates somewhat real tracks through the DTC if the energy of the particle is very high. The high energy makes it more likely for a straight track(a particle that does not change direction upon collision) to occur.

The data produced by this script is rigorous and only contains one hit per layer for every track and always produces the exact amount of tracks specified as a parameter. The data is structured using the format: Layer, Stave, Chip, x, y, eventID. This format is in a local perspective, as explained in section 3.3 which is also the data format that is retrieved from the readout units during a scan. A general source position for the particle beam is not taken into account by the script. A random point is chosen on the first layer, representing the beginning of a track.

To make the tracks seem more realistic, a Gaussian normal distribution is added to each hit, simulating the uneven path of a particle through materials(see Figure 3.1). This Gaussian distribution is reflected in the output data, where the calculated offset of an element rarely is the exact expected value but rather a value close to it. This is only true if a relatively small number of tracks is analyzed, as the offset is expected to get closer to the expected value as more tracks are analyzed.

The toy simulation script includes the possibility of introducing offsets in different elements of the detector. It makes it possible to see if the alignment algorithm works as expected and will be discussed further in chapter 4.

3.2.2 The Monte Carlo simulation

The Monte Carlo simulation used for this prototype was built using the GATE toolkit for medical physics applications. It is a GEANT4-based simulation platform used for emission tomography[33]. The platform is capable

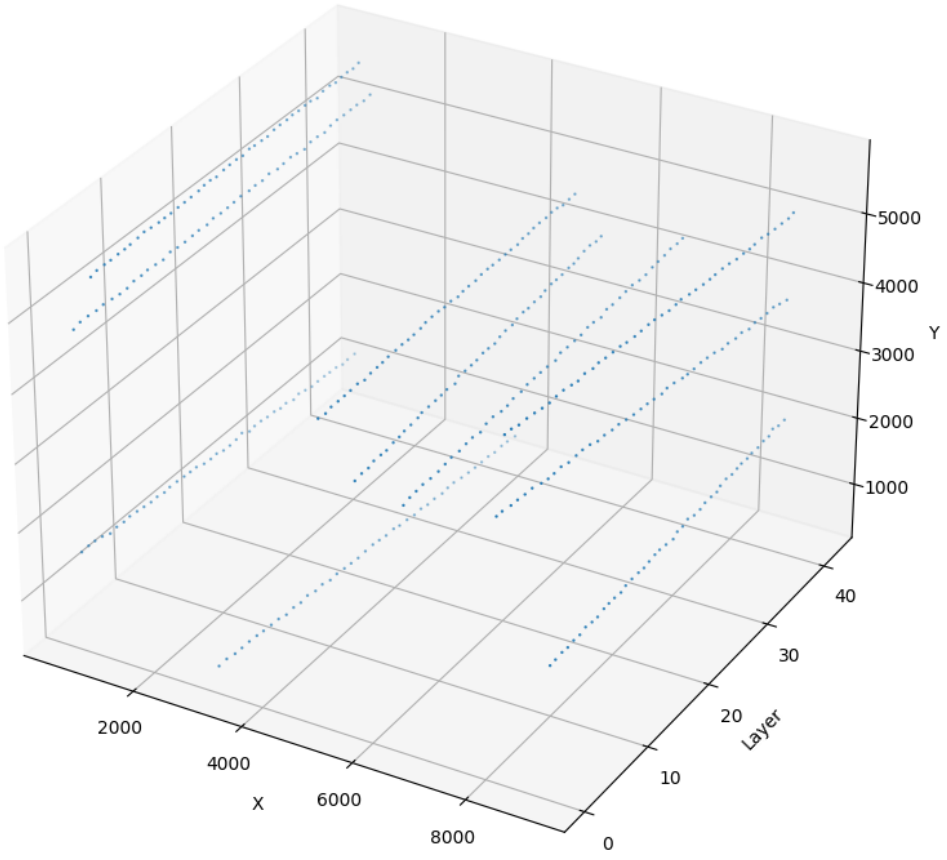


Figure 3.1: *10 tracks from the toy simulation script. Values are given as readout data in a global perspective.*

of simulating particles as they pass through matter. The GATE toolkit uses GEANT4 to create a set of C++ functions capable of defining volumes at a precise location and further do operations on these volumes like rotating them[34]. These volumes can be specified as several different materials that interact differently with particles.

This simulation script was not developed as part of this thesis but is critical for generating the data used by the alignment module.

In this case, the GATE toolkit is used to specify the geometry and materials of the DTC as well as the phantom used. This simulation is used to create as accurate as possible data for every part of the Bergen pCT prototype that needs to analyze detector data, such as track reconstruction, alignment, or general analysis of the functionality of the detector design itself.

The output data from this simulation is considerably more complex than that of the toy simulation described in section 3.2.1, as it describes everything happening with a particle when it is traversing the detector. This includes, e.g., directional angles and if a particle splits into other particles, where it also simulates what happens with these new particles. The large amount of data produced makes the output files consequently large, possibly several GBs depending on how many particles one simulates and how high the energy is. These file sizes are somewhat representative of actual data files and indicate the capabilities of the alignment module in terms of speed and data handling. The size of the output files are to a large extent related to the human readable format, which will be discussed further.

The two first layers of the DTC are called tracking layers. These two layers differ from the rest, both considering construction and usage. During the analysis of the data generated by the MC simulation, an issue with the first two layers was noticed. A slight "misalignment" was detected, making all calculations highly inaccurate. This was without any actual introduction of misalignment in the simulation. These two layers are removed when running the alignment algorithm to mitigate this "misalignment" of the first two layers. This is also why "layer 0" and "layer 1" are missing from the output when using MC data. The removal is done during filtering and is easily reversible.

Because of current limitations in the implementation of the MC simulation, a representative data set for alignment is difficult to generate. Data was

limited to the center of the detector without any offset. There are ongoing efforts to make data generation easier and have data generated for every element in the DTC, including the introduction of offsets.

3.2.3 Tools and Scripts

To begin working with the data generated by the DTC several scripts and tools were used. This made it easy to visualize the data and understand the general geometry of the detector and how it works. Some scripts were implemented in the early stage of this research and give insight into the presented work's evolution. They contain information about formats and methods used to produce a prototype and can be found in appendix A.

3.3 Coordinate Transformation

Because the DTC outputs a local format for each particle hit, a tool is needed to transform the local coordinate into a global perspective, describing every element in the coordinates of one unified detector geometry. Transformation of coordinates was necessary to begin the development of the alignment module and was therefore implemented early in the process. The coordinate transformer transforms a local detector data point into a global perspective. Local meaning the information about a hit on a single ALPIDE chip, and global being a hit in the perspective of the whole detector, independent of layer, stave, and chip values.

There are several coordinate transformation methods. Including going from local hits to global and vice versa, there is pixel-based transformation and millimeter-based transformation. These two base transformations are used in different objectives and have different accuracy, but both depend on the defined detector geometry. This detector geometry describes the distances between elements within the detector, e.g., the distance between two layers or the width and height of a ALPIDE chip as mentioned in section 3.4.3. In Figure 3.2 one can see a detailed explanation of the layer-by-layer geometry of the DTC with an explanation of all the materials used to construct it. We first see the tracking layers followed by the first calorimeter layer and a repeating pattern for all other layers.

The pixel-based transformation is, e.g., used to transform a global pixel value

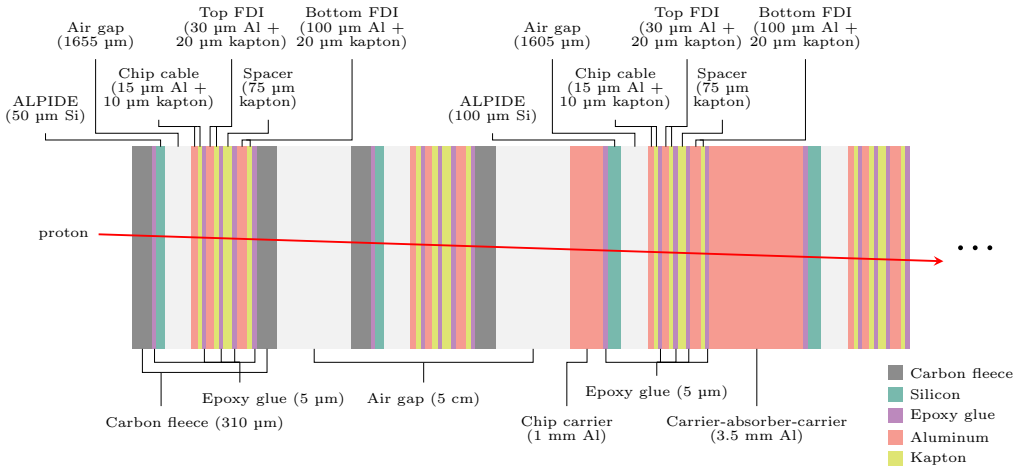


Figure 3.2: *A view of the DTC showing the size and material used for each layer. The figure is a up to date version of Figure 9 from [13] (Alexander Schilling, personal communication, May 5, 2022).*

into information about which element the hit was located. Knowing what element "owns" a specific offset is necessary to assign residuals to the right one. The millimeter-based transformation is used to get better accuracy for calculating offsets where pixel information is not accurate enough.

These two methods of coordinate transformation are likely to be used by other modules than the alignment module in the future of the pCT project. Therefore, it is essential that the coordinate transformer is optimized for time usage and does not hinder time-sensitive computations. This is also the case for alignment, as transformation is used in almost every part of the alignment algorithm. The general performance of the coordinate transformer will be discussed further in chapter 4.

In total, there are four implemented functions that provide different precision and functionality:

1. Local to Global pixel-based. This function takes local hit data and returns global pixel values x, y, z
2. Local to Global millimeter based. This function takes local hit data and returns global millimeter values x, y, z
3. Global to Local pixel-based. This function is given a global pixel value

and returns local hit value layer, stave, chip, x, y

4. Global to Local millimeter based. This function takes global millimeter data and returns local hit data layer, stave, chip, x, y

Before the Monte Carlo simulation data was introduced, the global to local millimeter-based transformation did not exist. This data is only given as global millimeter values, which is not the output format of the DTC data (given in local values). Therefore the global to local millimeter-based transformation had to be implemented explicitly for the MC simulation and may require modifications later as values change.

It is important to note that the coordinate transformer and the alignment module are two separate systems. Braking the coordinate transformer does not brake the alignment module. The alignment module does not affect the coordinate transformer whatsoever. Changing the output of the coordinate transformer one also changes the output of the alignment module as it is directly related to the global coordinates given by the transformer.

3.4 Implementing the Alignment module

The alignment module is part of the pCT-Online software and is responsible for producing a set of alignment parameters for elements inside the DTC. The module is built in three parts using a policy-based design mentioned in section 2.2.2, an input policy, a processing policy, and an output policy. The three parts each have one dedicated responsibility, making the design very modular and easy to change. This reduces the technical debt and the possibility that the module must be rewritten entirely.

3.4.1 Design

A set of requirements were defined for the prototype of the alignment module:

1. The calculation of offsets values from each element should be done in a reasonable amount of time.

2. Produced output should be stored in a format that can be easily used for further development

3. The module must be easy to maintain and service.

4. The module must be scalable as it is a prototype, and further improvements will be implemented.

All these requirements affected the final design. While all are important for the final product, not all were weighted equally. Maintainability, scalability, and the possibility for easy continued development had the most significant impact on the design. Other requirements will be heavier weighted as the development of the alignment module progresses.

Before any implementation began, the interface between the policies had to be defined. This is important to make sure each policy can be switched with any other policy that contains the proper functions. Figure 3.3 displays both the general objects/functions in each policy and with what format each function interchange information.

The input policy transforms the simulation data into tracks, and these tracks are passed to the alignment handler. The alignment handler passes each track into the offset producer and further into the offset finalizer. When done with all the tracks, the final output is sent to the output handler responsible for formatting and storing the output.

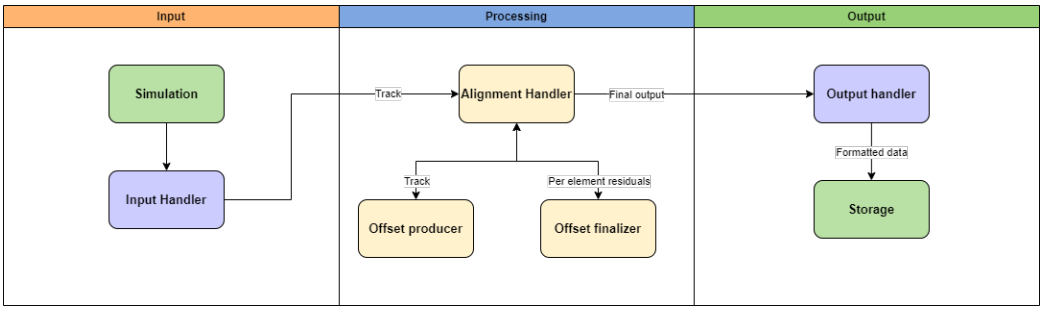


Figure 3.3: *The alignment module data flow.*

The format of the simulation data has yet to be specified, and the same holds for the final storage format. While the processing policy always works with the "track" format, "offset parameters" format, and "final output" format. The unspecified input and output formats for the input and output policy are excellent for flexibility and unnecessary as it is not crucial for the alignment algorithm to work. The used interface between the three policies is listed in Table 3.1.

Policy	Input	Output
Input Handler	Unspecified	AliTrack
Alignment Handler	AliTrack	unordered_map<...>
Output Handler	unordered_map<...>	Unspecified

Table 3.1: *Alignment module policy interface*

Here AliTrack is the defined data structure for tracks in the Alignment module. The **unordered_map** has the <key, value> pair

<staveKey , tuple<double,double,double,double,double,double>>

were **staveKey** refers to the stave element and the tuple being all six degrees of freedom for that element, explained further in section 3.4.4. **staveKey** uses a predefined hash function which is given as argument to the **unordered_map**.

To get a more specific idea of the structure of the alignment module, see Figure 3.4. Here we see the main executable "run-pct-alignment" and its dependencies. This is the central part of the alignment module where each of the "Handler" policies can be replaced or used interchangeably with other policy implementations.

3.4.2 The Alignment host

The interface of the alignment module is explicitly defined within the calls of the alignment host class. The host currently combines all the implemented policies into one working system. This class defines what functionality should be available in the policies, and if the functionality does not exist, it will throw an error message and abort processing. The host class knows nothing about the given policies upon execution, and if a called function does not exist within the given policy, it will stop when this function is called.

The host class is defined to include the three policies through template design as seen in Listing 3.1. It requires the three used policies to be defined upon the creation of the host class. An example of defining the alignment host class using toy simulation input can be found in Listing 3.2. A concrete definition and run of the alignment module can be found in Listing 3.3.

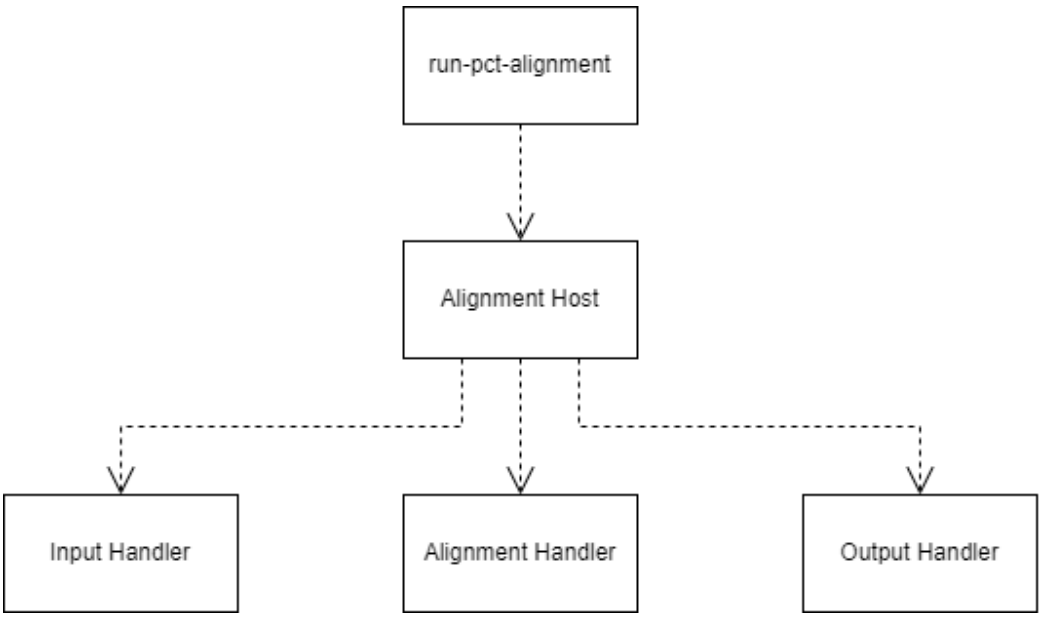


Figure 3.4: *The basic class structure of the alignment module.*

```
1 template<class InputPolicy, class ProcessingPolicy, class
      OutputPolicy>
```

Listing 3.1: *Alignment Host 3-tuple policy declaration.*

```
1 using AlignmentHost = AlignmentHost<AlignmentToySimInputHandler,
      AlignmentHandler, OutputHandler>;
```

Listing 3.2: *Defining Alignment Host with toy simulation input.*

```
1
2  //define used policies
3  using AlignmentHost = AlignmentHost<AlignmentToySimInputHandler,
       AlignmentHandler, OutputHandler>;
4
5  //create policy instances
6  auto inputHandler = AlignmentToySimInputHandler("path/to/input/
       data");
7
8  auto alignmentHandler = AlignmentHandler();
9
10 auto outputHandler = OutputHandler("path/to/output/file");
11
12 //create the host object with policy instances
13 AlignmentHost host = AlignmentHost(inputHandler,
       alignmentHandler, outputHandler, numTracks);
14
15 host.run();
```

Listing 3.3: *Creating a Alignment Host with toy simulation input and running it.*

3.4.3 Constructing Input Data

As described in section 1.2.1, the DTC contains many elements that are put together to create the final sensor. This concept of many small pixel grids (ALPIDEs) put together into a global perspective calls for a well-defined geometry. With geometry to work with, it is possible to transform a local hit (data from one ALPIDE) into a global hit position within the detector and vice versa.

There are many distances to define to have a complete specification of the geometry of the DTC. This detector geometry defines the distances between elements within the detector, e.g., the distance between two layers or the width and height of an ALPIDE chip. The definitions are created with the C++ data structure "struct," and the code is implemented with the possibility of having a user-defined geometry. Every part of the code that relies on geometry is not strictly relying on any specific value.

A general data structure for the particle tracks has been created from the definition of the geometry. This structure contains all relevant data points

from each hit from every layer, ultimately representing a particle track. As explained in section 1.2.1, the detector is built up of 43 layers, each layer containing 12 staves and each stave having nine ALPIDE chips. Based on this, the data structure for a hit can be represented like in Listing 3.4 and a track can be represented like in Listing 3.5. As seen, a track is a vector of AliHits.

```
1 struct AliHit {
2   int layer;
3   int stave;
4   int chip;
5   int x;
6   int y;
7 };
```

Listing 3.4: *C++ representation of alignment hit data*

```
1 struct AliTrack {
2   std::vector<AliHit> hits;
3 };
```

Listing 3.5: *C++ representation of alignment track data*

The AliHit structure contains the relevant data points for each hit. What layer the hit is from, which stave on that layer, the chip from the stave, and the local x,y (sometimes referred to as column and row) coordinate of the hit. Regarding the element rotations mentioned in section 1.2.1, it is now possible to represent every hit from a global perspective for analytical calculations and from a local perspective to know the origin of the hit.

The objective of the input policy is to read data from a file and structure the data as AliHit and AliTrack. At this stage, the data is verified, and if any problems occur, the run is aborted. As long as this is done correctly, the processing policy can use the data without problems. If the input file is large, containing millions of tracks, one can decide to give a specific amount of tracks **n** to be analyzed. **n** is given as a parameter to the input policy where it stops reading after **n** tracks. Limiting the number of tracks can help with reducing the run-time in a case where one knows there is a diminishing return after **n** tracks have been analyzed.

3.4.4 Calculating the Alignment parameters

To make sure the readout data from the DTC has as few errors as possible from the unavoidable misalignment mentioned in section 1.2.2, the exact rotational and directional values of each element has to be corrected. Therefore knowing what these alignment constants are is key to producing a prototype.

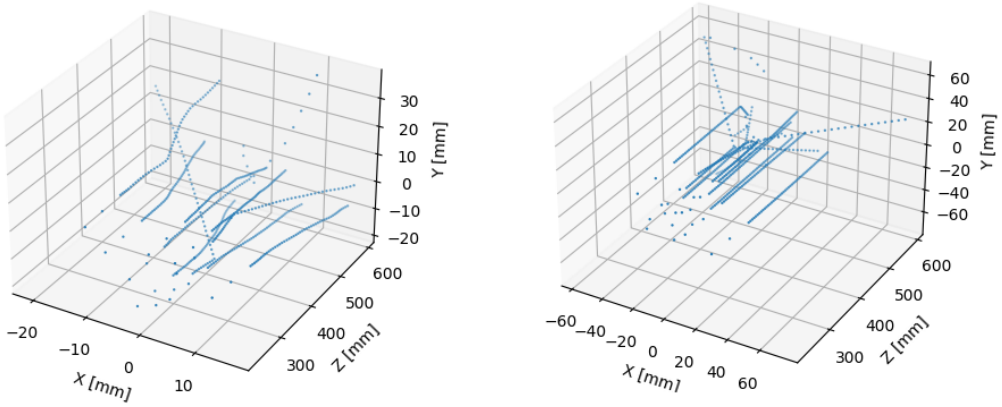
For the directional alignment constants, knowing of the three-dimensional structure of the DTC, it is clear that the elements can shift in all x, y, and z directions. As explained, the DTC is layer-based, which means that each layer can have three other misalignment parameters, namely (yaw) rotation around the z-axis and (pitch) rotation around the y-axis, and (roll) rotation around the x-axis. Optional alignment parameters will be discussed further in section 4.3. To summarize, each element that is considered to be aligned needs six different alignment constants, which can be structured like this:

(x, y, z, yaw, pitch, roll)

As will be explained in section 3.4.5, the stave element has the most focus during the development of this prototype as this element is most likely to have individual directional shifts. For the rotational shifts, all elements of an entire layer are expected to have the same values, which should be researched in further iterations of the module. As it is more difficult to produce some of these parameters than others, the x and y parameters have the most focus for this prototype. While x and y offsets can be produced through histogramming, more complex methods are needed for the four other parameters.

The job of the processing policy, also known as the alignment handler, is to calculate all six alignment constants based on the input it receives from the input policy. To do alignment, we assume some things always to be true. The assumption does not mean they are not considered within the system, only that the alignment handler does not do any calculation to know if it is true or not. Most apparent is the concept of straight particle paths. A particle that changes direction during travel would make linear regression produce incorrect values. An example of these straight tracks can be seen in Figure 3.5, where you also can see the difference between low energy unusable and high energy usable tracks.

Using linear regression (OLS) to calculate the expected track of a single particle, based on the actual values of that particle, and doing this for all



(a) (b)

Figure 3.5: *Figure (a) shows 10 tracks from protons at 500 MeV with a large change in direction. Figure (b) shows 10 tracks from protons at 2500 MeV with mostly straight tracks. Z-axis values are given in relation to where the detector is located in the simulation space. Zero indicates the center of the detector.*

the recorded particles produces large numbers of residuals. A residual is the deviation between the predicted track(from the regression model) and the actual track and can be directly used to find if an element is offset in some direction. Gathering the residuals from every track for all elements, one can take the average of all the residuals(using histograms) and use this as the offset value for each element, see Figure 3.6. The method is based on the law of large numbers, which states that by performing the same experiment a large number of times, the average value tends to become closer to the expected value the more trials that are performed[35]. It also explains that alignment cannot be done with a small number of tracks, explained further in section 4.1.4.

As explained in section 3.4.3 the data given to the alignment handler is in the form of local pixel data (see Listing 3.4). To both increase accuracy and make the data usable, the data is transformed into global millimeter "hits" described in section 3.3. Transformation makes sure the structuring of the ALPIDEs described in section 1.2.1 does not influence the linear regression. A pixel-based transformation is done alongside the millimeter-based one to

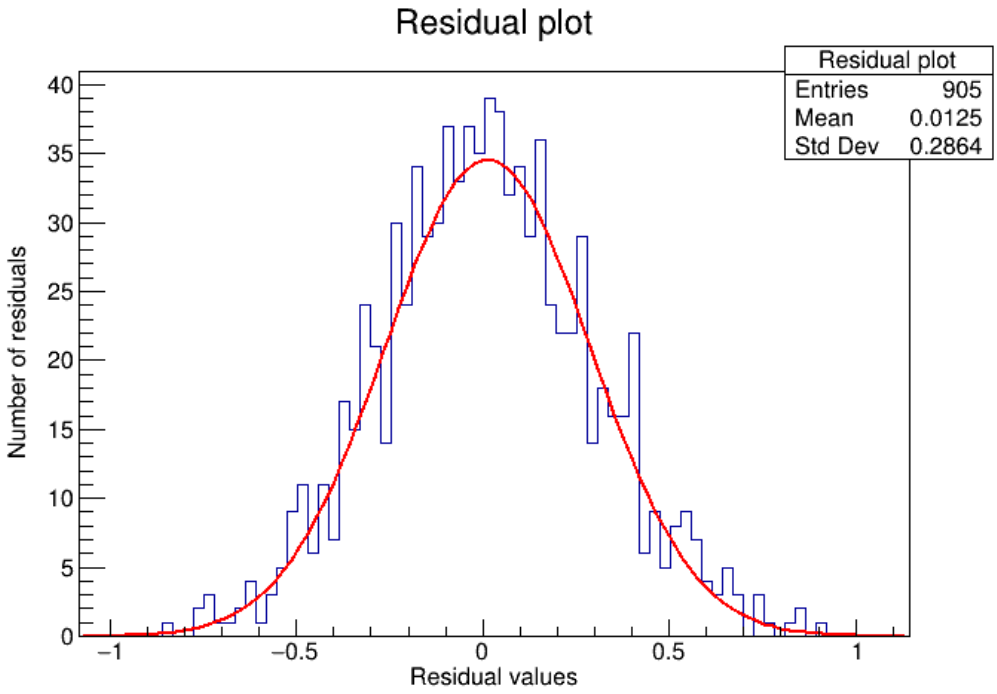


Figure 3.6: *The combined residuals of a DTC element in a histogram displaying the calculated offset for that element as mean. Residual values are given in millimeters.*

find what element the current residual value belongs to based on the predicted track. This is why the regression model is implemented using generics. Because of this, the same class implementation can do regression on both data types. To mitigate some confusion, the implementation of linear regression is done using concepts from machine learning like "fit" and "predict," meaning calculate regression line and extract prediction, respectively.

The output of the alignment handler is given as a list that contains the six alignment constants for each element. Algorithm 1 contains the proposed algorithm used for calculating the offset parameters as pseudo-code. See Table 3.2 for an explanation of the used variables.

Variable	Explanation
t	Currently parsed track already converted to global coordinates.
T	Collection of all usable tracks from input.
p	Predicted track based on information from **t**.
e	Element hit by predicted track(as index) for each residual **r**.
E	All elements in DTC.
r	Residual of singular hit from track(**t**).
r'	Collection of residuals from all hits in track(**t**).
R	Collected residuals for current element **e**.
w	Calculated offset for element **e**.

Table 3.2: *Explanation of used variables in Algorithm 1*

Algorithm 1 Algorithm for calculating alignment offsets.

```
1: //Note this is done separately for x and y axis
2: for t in T do
3:    p = predicted_track_using_OLS(t)
4:    r' = residuals of p given t
5:
6:    for e in elements of p do
7:       r = r'[e]
8:       add r to R[e]
9:
10: for e in E do
11:    w = mean(R[e])
```

3.4.5 Formatting and Storage of the Output data

JavaScript Object Notation or JSON is a syntax used for structuring data into text-based objects. It is derived from the ECMAScript programming language but is used independently of any programming language. Using a set of structuring rules, data can be structured and represented in a text format[36].

The JSON syntax was created to be easy to read and write for humans and machines. It is based on a subset of the JavaScript programming language standard ECMA-262 3rd edition - December 1999. JSON is a popular data-interchange language that is built on two structures that are universal for nearly all programming languages[37]:

1. A collection of name/value pairs. In languages referred to as objects, records, structs, dictionaries, hash tables, keyed lists, or associative arrays.

2. An ordered list of values. In languages referred to as arrays, vectors, lists, or sequences.

The most basic empty JSON object is simply two braces "{}." To make any more complicated JSON object, the standard defines it as in Figure 3.7. Here **value** is a standard definition for any string, number, object, array, true, false, or null.

JSON is used in the alignment module to store output data. The format

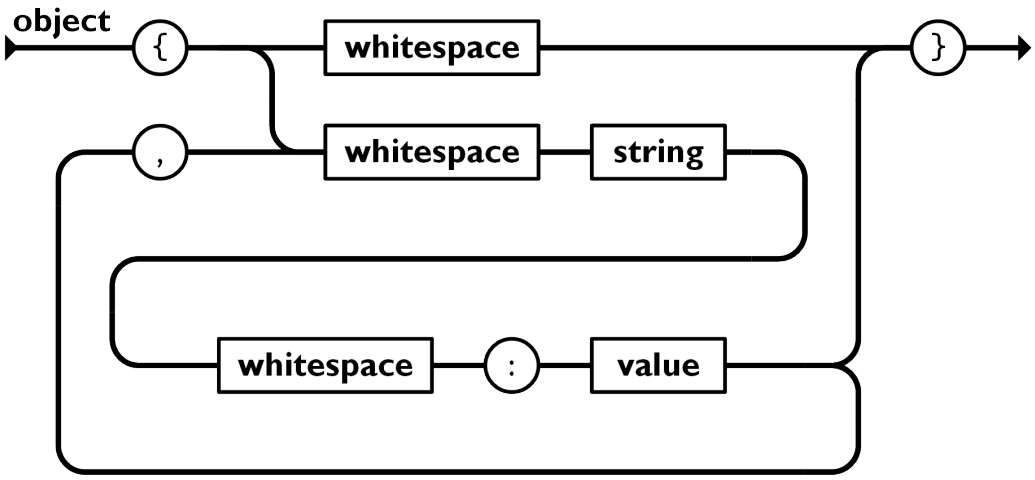


Figure 3.7: *The JSON object representation [37].*

makes it easy for anyone to use the data later. It makes sure the data is as accessible as possible for further development, whether in C++ or any other language. Here the LAYER, STAVE(**staveKey**) key of the alignment handler output is used to structure the JSON objects for the final output. The JSON structure of the output file is as in Listing 3.6.

```
{
    "Layer 0":
    {
        "Stave 0" :
        {
            "x": number,
            "y": number,
            ...
        },
        ...
    },
    ...
    "Layer 42" :
    {
        "Stave 0" :
        {
            "x": number,
            "y": number,
            ...
        },
```

```
21              ...
22          }
23  }
```

Listing 3.6: *Alignment output JSON object structure.*

The order of the layers and staves within the JSON object is not strict as, e.g., "Layer 10" is less than "Layer 2" (because strings are compared character by character), and therefore "Layer 10" is stored before "Layer 2". This example happens multiple times in the output JSON object. However, it should not affect the usage of the object within any programming language. The example in Listing 3.6 is one where staves are the focused alignment elements. It is a decision made because staves are the smallest detector element that still has a high potential for noticeable shifts. The focused element is not a vital part of the code, and it is relatively easy to change the focused element between layers, staves, and chips. To change the focused element, change the output key as defined in Table 3.3, and respectively change what residual values are combined to create the final offset values.

Element	**Key name**	**Key value**
Chip	chipKey	tuple$< int, int, int >$
Stave	staveKey	tuple$< int, int >$
Layer	layerKey	int

Table 3.3: *Defined element keys.*

3.4.6 The Working Alignment module

The currently implemented functionality of the alignment module covers everything mentioned in section 3.4.1. Using the predefined interface, it produces an output file storing the calculated offset parameter for every element. It can process sizeable MC input files, with thousands of tracks, in seconds. For now, this is done using the .csv format with lots of redundant data, making benchmarking inaccurate. There are plans for binary input files containing only necessary variables to fix this issue. A timer is included to time each execution of the module, making it easy to see the difference between current and new implementations. Figure 3.8 illustrated the alignment module and its policies.

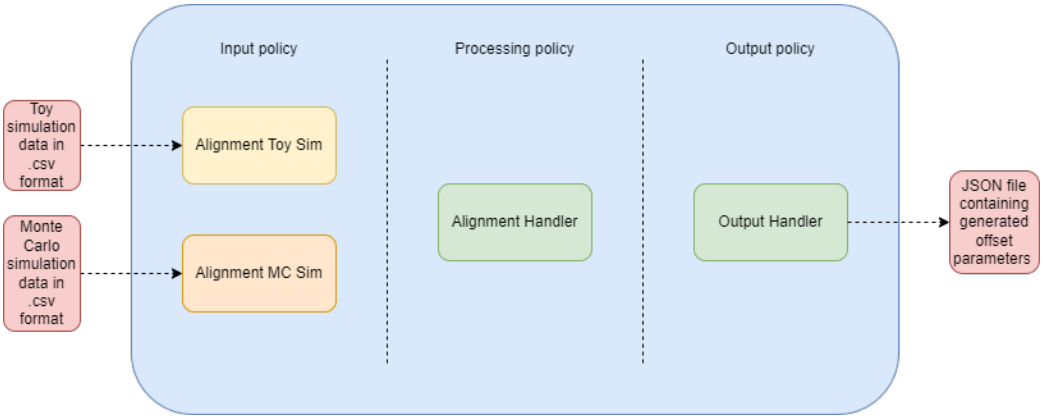


Figure 3.8: *Overview of the current software modules. All modules visualized here are new and implemented for use in the alignment module.*

Currently, two different input policies are implemented and used, one for the toy simulation and one for the Monte Carlo simulation. The use of each policy is defined through commands and injected based on given parameters. The host class (see Figure 3.4) is responsible for upholding the defined interface, which allows for using different policies where needed.

Chapter 4

Analysis and Assessment

In this section, several experiments will be explained alongside the results from these experiments. Issues and limitations with the implementation will also be discussed. The methods used for developing the alignment module are discussed, ending with a summary of the impact of this thesis.

4.1 Alignment performance

As mentioned in section 2.1.2, running a "blind" test of the alignment module would indicate if it works as expected. While visualizing the data through scatterplots does not show the algorithm's accuracy, comparing the output value to the value given as input to the simulation does. The accuracy is related to the number of analyzed tracks, which will be discussed further in this chapter.

A limitation introduced from the available Monte Carlo input data is the size of the files. It is not a problem for the Alignment module but rather a local workspace problem as hard drives fill up. While testing, the files had to be limited to around 20 GB, reducing the number of available tracks. The limited filesizes may be a reason for unnoticed faults or incorrect accuracies.

4.1.1 Accuracy using toy simulation data

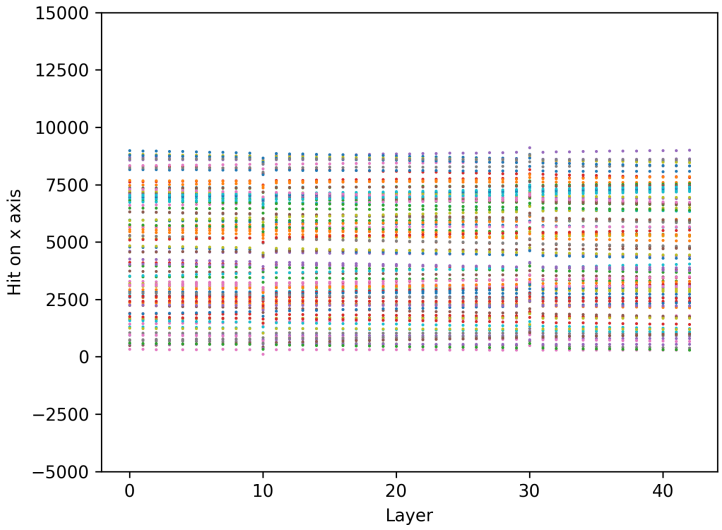
To determine the accuracy of the alignment module, comparing the results from the alignment module to the known values will indicate if it is accurate or not. While interference from noise in the input data is difficult to counteract in the results, results are still expected to be consistently equal to the expected value. Approaching this analysis with a basic example can be valuable for further development and comparison to further results. Using the toy simulation to apply offsets at layers 10 and 30, the resulting offset values can be compared to the applied offsets. For this basic example, a 200 pixels shift in both the x and y direction for layers 10 and 30 was applied. This is visualised in Figure 4.1.

Using the output from the alignment module on 50000 tracks, one can see if the result is as expected. As a start, let us consider the total offset applied to layers 10 and 30. The applied offset is equal to 200 pixels where each pixel is of size 29 µm, which translates to 0.029 mm. This means the total shift is expected to be $200 \times 0.029\,\text{mm} = 5.8\,\text{mm}$. Comparing this to the output values, it is almost correct, see Figure 4.2. The exact values from the output were x = 5.66 mm and y = 5.66 mm.

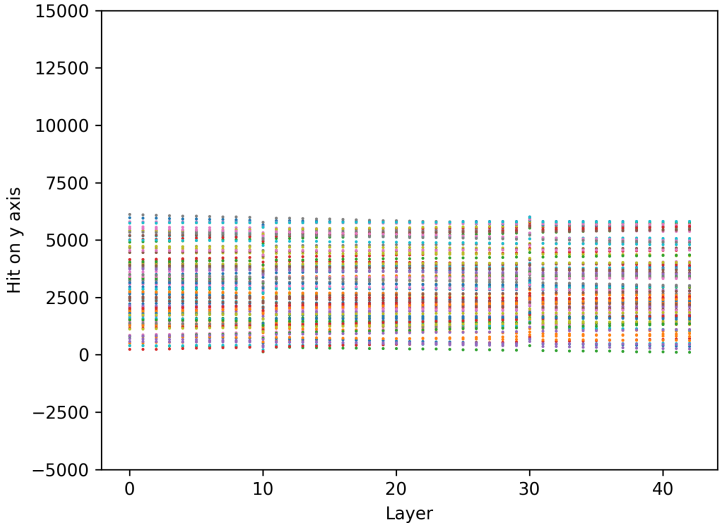
Keep in mind that this kind of large misalignment is uncommon. A more realistic scenario would be small offsets on every layer, while some elements may have larger offsets. The used regression model is not robust enough for this, which will be discussed further in section 4.3. Two things to be aware of for these data:

- Large misalignments like this (5.8mm) example can be corrected using mechanical methods and should not be possible because of the structure of the DTC.
- The method used for this prototype is known to work on "large" misalignments. However, if several tiny misalignments are present in every layer, more complex methods need to be implemented to get a precise output.

The example introduced here is used to prove the principle of accurately detecting misalignments. The 0.2 mm inaccuracy seen would tell us it is not very accurate, but in fact, it is because of limitations of the used regression model. The regression limitations and the case of implementing a more



(a)



(b)

Figure 4.1: *(a) showing a top down view of 100 tracks in the DTC with offsets at layer 10 and 30. (b) showing a side view of the DTC with the same offsets and tracks as in (a). Values are given as readout data in a global perspective.*

complex algorithm will be discussed further.

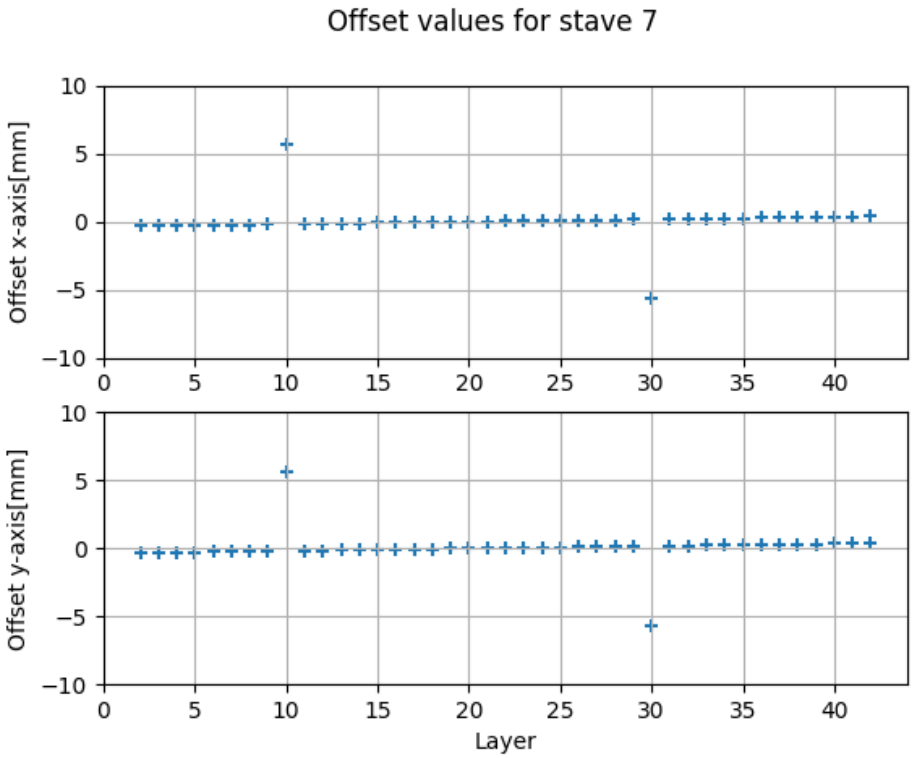


Figure 4.2: *Per layer offset values produced by data from the toy simulation. Notice how the introduced misalignment creates offsets on every layer.*

Now we know it is possible to get accurate measurements by analyzing the track data. Seeing how it performs on more complex Monte Carlo simulations with misalignment is still not possible. However, performance on perfectly aligned elements will still be analyzed in section 4.1.2. An illustration of how tracks are realigned using the output data is available in appendix B.

4.1.2 Accuracy using MC simulation data

When using Monte Carlo simulation, we know the data is very likely to be reproduced during real-world usage of the DTC. This is why the results gathered from MC simulation data are much more interesting than the toy simulation. However, the state of the simulation does still not allow for

the introduction of misalignment. This is why the expected results from this data should be much less than the width and height of a pixel while expecting zero is unrealistic because of noise in the data. Table 4.1 shows some selected offset values produced by the alignment module on Monte Carlo data without misaligned elements.

Layer	Stave	Offset X	Offset Y
11	4	$-0.221\,\mu m$	$0.206\,\mu m$
13	5	$-0.683\,\mu m$	$-0.463\,\mu m$
15	7	$0.375\,\mu m$	$3.448\,\mu m$
17	4	$-0.727\,\mu m$	$-3.512\,\mu m$
21	4	$-0.458\,\mu m$	$-5.085\,\mu m$
24	4	$-0.957\,\mu m$	$-5.513\,\mu m$
30	5	$0.153\,\mu m$	$-0.709\,\mu m$
35	6	$-0.280\,\mu m$	$0.602\,\mu m$
39	4	$-0.227\,\mu m$	$0.619\,\mu m$
42	5	$-0.155\,\mu m$	$0.554\,\mu m$

Table 4.1: *Offset values from MC simulation data.*

Because the data is generated with very high energy particles (10000 MeV), every parsed track has a relatively straight path with little to no angle. The output data is therefore expected to be around zero for every element. On some staves, the offset is observed as exactly zero (0.000000). Figure 4.3 shows every offset value on each layer from stave 7.

4.1.3 Multiple Layers Offset using Toy Simulation

An example with several layers being offset was performed to understand the alignment module's accuracy better. With this example, a layer-based accuracy comparison can be made. To do this correctly, a large number of tracks were simulated, which makes sure each element had a sufficient amount of tracks to do calculations. The values will always be better with more tracks, but at one point, the number of tracks becomes too many to be considered realistic. This is why this example uses a total of 1.000.000(one million) tracks. The reason why so many tracks are needed will be explained further in section 4.1.4. The predicted offset for the x and y-axis is listed in Table 4.2 together with the actual offset given in millimeters. The difference

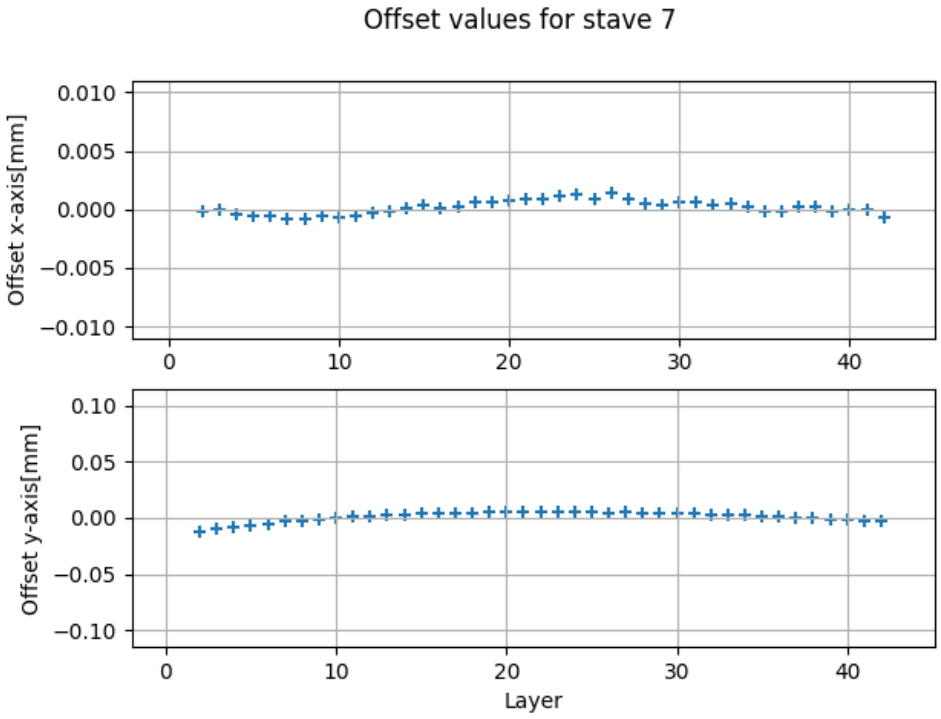


Figure 4.3: *Per layer offset values produced by data from the MC simulation given in millimeters.*

between prediction and actual offset is also given. This is also visualized in Figure 4.4.

Layer	Offset X	Offset Y	Pred X	Pred Y	Diff x	Diff y
6	0.2924	-0.17544	0.29256	-0.18160	-0.00016	0.00616
9	-0.11696	0.20468	-0.11712	0.20038	0.00016	0.00429
10	0.08772	-0.05848	0.08789	-0.06238	-0.00017	0.00390
11	-0.2924	-0.05848	-0.29215	-0.06136	0.00024	0.00288
15	0	0.26316	0.00068	0.26414	-0.00068	-0.00098
22	0.23392	0.08772	0.23568	0.09187	-0.00176	-0.00415
23	-0.23392	-0.08772	-0.23202	-0.08268	-0.00189	0.00503
30	-0.17544	-0.1462	-0.17328	-0.13766	-0.00215	-0.00853
35	0.20468	0.1462	0.20848	0.15630	-0.0038	0.01010
40	-0.05848	-0.23392	-0.05461	-0.22141	-0.00386	-0.01250

Table 4.2: *Predicted offset values for all misaligned elements in multi layer offset example. Offset represents the introduced misalignment, pred represents the predicted offset, and diff is the difference between offset and pred for each axis. Note: Some unnecessary precision has been removed. All values are given in millimeters.*

Every offset prediction is far less than the width of a pixel, which can be considered a reasonably good accuracy. It proves that the alignment module can predict misalignment values of several elements at once. In the case of the current implementation this relates to staves as the corrected element. Using even more tracks would further improve accuracy. The data is comparable to the data displayed in section 4.1.1 which means this method of offset calculation does not depend on the number of misaligned elements.

An issue noticed during testing is the case where several elements are shifted in the same direction. It leads to a systematic skew in all elements, reducing accuracy. This issue is also present in the data presented in section 4.1.1 where the output is off by 0.2 mm. A possible solution to this problem is to use a different regression model, discussed further in section 4.3.

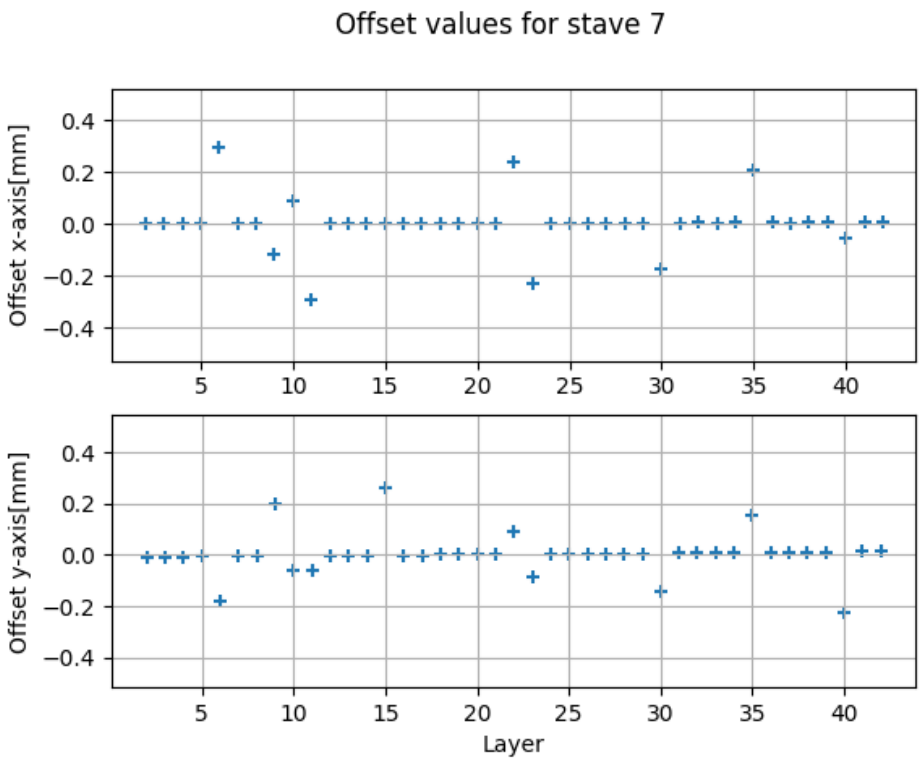


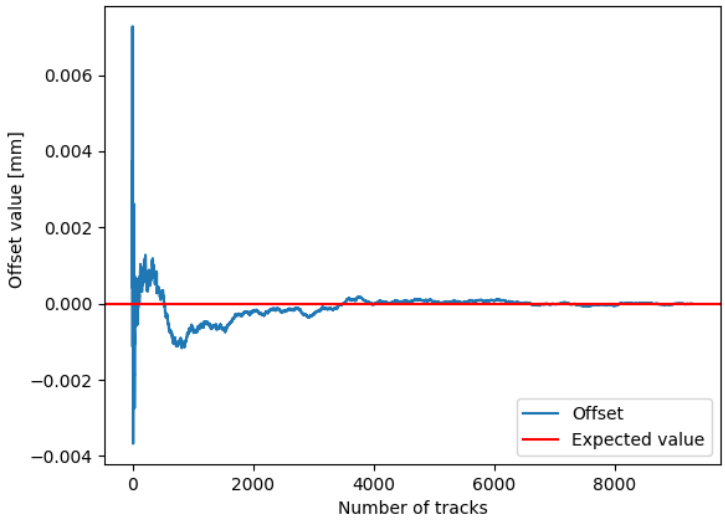
Figure 4.4: *Offset values per layer for stave 7 in the multi-layer offset experiment.*

4.1.4 Convergence

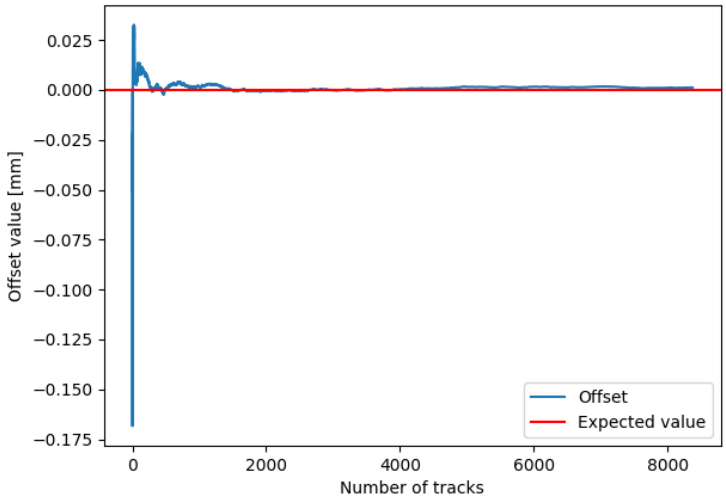
Calculating the offset for each track analyzed indicates the necessary amount of tracks needed per element to get an accurate offset value. To test this, measurements were done on a random but same element for both the toy and Monte Carlo simulation on the x-axis and can be found plotted in Figure 4.5.

It shows that after a certain amount of tracks, the offset value converges towards the expected misalignment value of that element. While the Monte Carlo simulation seems to get a much better result than the toy simulation, both have an accuracy below a single pixel's width. Keep in mind that the plotted data is in millimeters, and the toy simulation is expected to have less accurate and more noisy data.

The convergence of the offset values could be used to find a limit to the



(a)



(b)

Figure 4.5: *Per track offset values in millimeter produced by data from the MC simulation(a) and toy simulation(b).*

number of analyzed tracks to reduce the run-time and complexity of the alignment module. The current state of the alignment module is likely to be changed as more complex methods are introduced. However, by the values presented in Figure 4.5 we see that after about 4000-6000 tracks, it converged both in the toy and MC simulation scenario. This indicates a minimum number of tracks needed per element for this alignment method.

The plotted data also indicates the danger of having too few tracks passing through an element. Too few tracks may lead to false offset values for that element. Filtration should be done when producing the final output of the Alignment Handler policy to remove offset values calculated from too few tracks.

4.2 Benchmarks

4.2.1 The Coordinate Transformer

The coordinate transformer is not exclusively made for the alignment module but rather as a tool needed to produce usable data for many data analyzing modules within the system. This transformation class may also be used to transform coordinates for the real-time full data stream of the readout chain. Because the readout chain is very time-sensitive, every amount of time added must be considered. Hence benchmarking the transformer to find its potential throughput is beneficial.

Table 4.3 displays the speed of each of the four transformation functions explained in section 3.3. Iterations specify the number of iterations of the function that were done to get a stable result.

Function	Timing	Iterations
localToGlobalHitCoordinatePixel	13.1 ns	56000000
localToGlobalHitCoordinateMM	16.3 ns	40727273
globalToLocalHitCoordinatePixel	49.8 ns	14451613
globalToLocalHitCoordinateMM	17.6 ns	37333333

Table 4.3: *Time usage of all coordinate transformation functions.*

This timing is done using the google benchmark library[38]. The processor specifications were (24 X 3701 MHz CPU s), which may be a faster processor than what will be used, but in general, one can expect transformation to be done in some nanoseconds. Using the above data, the expected throughput of the coordinate transformer is:

1. Local to Global pixel based: 1s/13.1ns = 76MHz transformations per second. Given in tracks per second 76MHz/43 = 1,7MHz.
2. Local to Global millimeter based: 1s/16.3ns = 61MHz transformations per second. Given in tracks per second 61MHz/43 = 1,4MHz.
3. Global to Local pixel based: 1s/49.8ns = 20MHz transformations per second. Given in tracks per second 20MHz/43 = 0.5MHz.
4. Global to Local millimeter based: 1s/17.6ns = 57MHz transformations per second. Given in tracks per second 57MHz/43 = 1,3MHz.

Given that only the Local to Global conversion has to be done in real-time, it can theoretically handle the data generated by 76,335,877 pixel activations per second when pixel-based. When millimeter-based, it can handle 61,349,693 pixel activations per second.

The current implementation uses a basic arithmetic approach to the problem to optimize for speed. Not using any object-based structure to do transformation makes the implementation complicated and less maintainable. As this is a prototype, it is expected that more work has to be done before this is used in real scenarios. It would be beneficial to do more research on implementing a more maintainable coordinate transformer.

4.2.2 The Alignment Handler

Even though the alignment handler has several helper functions, only one function is relevant to benchmark for this policy. This is the **analyseTrackXY()** function responsible for calculating offset parameters from a single track. The performance of this function is relevant for the time required to analyze **n** amount of tracks. The speed is listed in Table 4.4.

Function	Timing	Iterations
analyzeTrackXY	8545 ns	89600

Table 4.4: *Time usage of Alignment Handler functions.*

The speed is roughly the same as 117027 tracks every second. Because of this, one would expect the alignment module to use only some seconds to calculate enough tracks, but this is not the case. The primary source of time used for the alignment module is the Input handler which will be discussed further in section 4.3.7. The alignment handler's fast speeds result from the used method for calculating offsets as it is not computationally heavy. Other alignment methods require heavier computations, resulting in higher time usage, which will be discussed further.

4.3 Method analysis

4.3.1 Policy Pattern

Using the policy pattern ensures that the implemented alignment module upholds the modularity of the pCT-Online package. The pattern has made development more straightforward and the code-base easier to understand. Based on the fact that this is a prototype, the primary outcome of using this pattern is the possibility for further development on this implementation without any significant refactoring. As mentioned in section 3.4.4 a more complex method for calculating all the offset parameters is needed, and this pattern should make the process of implementing this easier.

A significant drawback of using this pattern is the problem of documenting the implemented interface as the module becomes complex. The interface has no explicit representation in code and therefore has to be well documented.

4.3.2 Data storage

As an output file format, JSON is excellent for this task. It is widely used and easy to read for both humans and machines. Other options could have been used, like XML or CSV, but the easiest to use and implement is still JSON. This option was also chosen because of its resemblance to non-relational database storage. Further development of this module will include a database for the output data. This way, timing, and other data points can be generated together with a history of misalignments for each element.

A database like MongoDB uses the JSON format to store data, making it a relevant database for storing alignment output. At this point, the future of data storage for the alignment module is not decided. Therefore choosing the most frequently used and most accessible format was favorable.

4.3.3 Offset calculation method

The current method for calculating the offset parameters can produce reasonably accurate calibration parameters for the alignment in x- and y-direction. This is shown in section 4.1.1 and 4.1.3. To calculate the other alignment parameters, a more complex method is needed. Some algorithms for this have already been developed for other particle detectors and require good

knowledge of linear algebra. It also requires one to understand the details of how tracks behave inside the detector. This will be discussed further in section 6.

The current implementation can perform pre-alignment to correct for shifts in the x and y direction before the alignment of rotations is considered. It is another benefit of using the policy pattern, allowing one to first run a pre-alignment before applying the more complex policy for a detailed alignment.

4.3.4 Using simulation data

Implementing a simulation script gave great insight into the data format generated from the DTC. It has been crucial for the possibility to implement the alignment module and the coordinate transformer. The format has already been covered in section 1.2 and 3.4.3.

The main reason simulated data was used in this thesis is that accurate misalignment has to be introduced to the data. Introducing this misalignment in a real detector is not possible. It made analyzing the accuracy of the alignment module possible because the location of the misaligned elements was known. It also made it possible to prove that this form of alignment is possible.

The limitations in the number of tracks analyzed because of the large input files are expected to be resolved. There are plans to remove all unnecessary data points and make the input stream binary. It will allow for a higher density of usable tracks within the input file.

4.3.5 Regression model

During testing, it was found that the OLS regression model relies on a fairly even spread in the offset directions on each layer. Too much shift in one direction will introduce a systematic shift of all offset values. The systematic shift is not optimal if some elements have no shift. It was discussed in section 4.1.3 and is visualized in Figure 4.2. Another well-defined example of this systematic shift is shown in Figure 4.6, where misalignment(2 pixels) in the same direction from just a few elements creates offsets in every element.

While the robustness of the used regression model is not too relevant when the offsets are smaller, it is still an error that is the source of less accurate

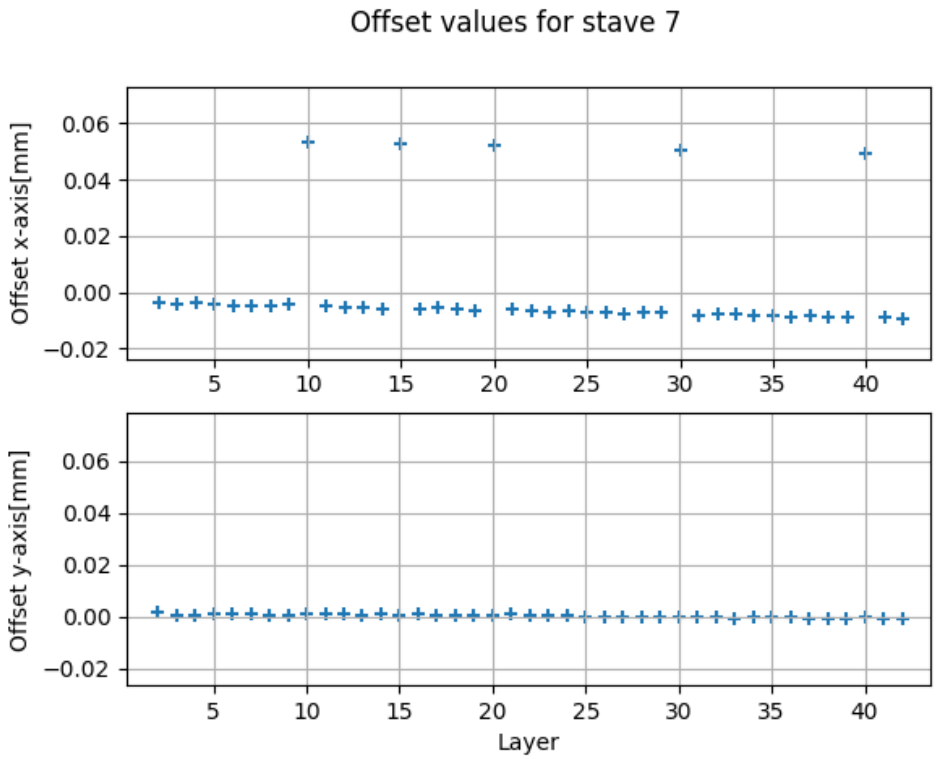


Figure 4.6: *Visualization of how all parameter values are skewed from a few outliers. All values are expected to be lined up at zero, but offsets from a few elements create a offset in every correctly aligned element.*

offset values. A more robust linear regression model could be used to improve this. A new model must be aware of outliers in the given data, making every large offset unnoticeable in the regression function, also known as the predicted track. Because residuals can be affected by the misalignment of any other element, implementing a more robust model could remove the issue as outliers will no longer skew the prediction.

4.3.6 Optional Alignment parameters

The general concept of alignment is to find where an element is located relative to the other elements. It is done by analyzing the data acquired by the DTC. The data acquired are only represented by x and y coordinates

from each layer. It means that the only correction that can be done directly is to each hit's x and y coordinates. There are no z-axis or rotations to adjust. This leads to several different ways one can define the alignment parameters. For this prototype, the idea was to find the misalignment values of each element, not correcting any tracks. The correction has to be done later. As this was the idea, giving the offset in millimeters for each element for the six parameters mentioned earlier was done. Another option could be to combine some of the misalignment parameters, e.g., the shift in the z-axis could be defined as $\frac{\Delta x}{\Delta z}$. This option says how much x shifts based on the shift in the z-axis. And the same for y, $\frac{\Delta y}{\Delta z}$. This alteration would change the alignment parameters to something like:

(x, y, $\frac{\Delta x}{\Delta z}$, $\frac{\Delta y}{\Delta z}$, rotations)

Optional parameters may be researched in further work on the alignment module to optimize accuracy when applying the parameters to realign elements.

4.3.7 Profiling

To get an insight into the performance of all parts of the alignment module, a tool called Valgrind was used to profile the code. More specifically, the Callgrind tool. Callgrind is a profiling tool that records the call history among functions in a program's execution[3]. It gives a very detailed rundown of every function and how much relative time is used within each one. The data can be utilized to find where optimization is needed and where other improvements can be made.

Where the current implementation uses most of its time is in the **getTrack()** function of the input policy. The function is responsible for fetching a track from an input data file. This behavior is expected as the input file is large and data filtering is done here. This function does not have a consistent runtime because of the unfiltered data. Not accounting for the track fetching function, the system performs remarkably well, using only seconds to parse hundred-thousands of tracks.

4.3.8 Testing

Even though test-driven development rules have not been followed, unit tests have been implemented to ensure the proper functioning of the implemented code. As of now, automatic tests cover the functionality of the coordinate transformer from local to global and vice versa for pixel and millimeter-based transformation.

The alignment module functionality has been developed through manual testing. These tests are undocumented and do not ensure exact functionality. Observing the output data, we can find if it is functioning as it should. E.g., if we know tracks are passing every stave in the detector, we should expect the output to contain $12 * 43 = 516$ elements. Any other results would indicate that some functionality is not working correctly. Discrepancies in the output have not been noticed with the current implementation with several million tracks tested.

4.4 Impact of thesis

The work done in this thesis is based on work done by many others but is also a completely new addition to the Bergen pCT project. At the beginning of this work, a general way to transform local particle hit data into a global perspective was implemented. The new global data was then used to produce an alignment module capable of producing misalignment values for elements inside the DTC. The data transformation is expected to be used by many other future projects as actual data becomes available from the DTC. The software developed is a solid prototype highlighting the key concepts needed for alignment of the DTC elements and what is needed from this module in the future.

Chapter 5

Conclusion

In this section brief explanation of the performance of all the included implementations will be discussed. The results will be summarised, mentioning both issues and what was done right.

5.1 Performance Evaluation

5.1.1 The Coordinate Transformer performance

The data presented in section 4.2.1 show that the coordinate transformer can potentially transform millions of hits per second. As this can potentially be run in parallel for each layer it results in $43 \times 61,349,693 = 2,638,036,799$ transformations every second. However, testing has to be done to ensure this does not introduce any errors to the data stream processing.

The implemented coordinate transformer allows estimation of the required time for transformation. It is based on trivial operations and there is only little potential for optimization. Hence it is a statement on the feasibility of real-time processing.

5.1.2 The Alignment Module performance

As explained, the performance of the alignment module is based on how accurately it can find the offset values of each element inside the DTC. It is validated by running a "blind" test where the offset values are known to

us but not known for the alignment module. Suppose it produces values representing the actual offset, then we know that it works as intended. This was shown to be true in section 4.1.

The implemented software is capable of producing alignment offsets and is a solid foundation for further development of the module. Except for the errors produced by the chosen regression model, the ability to produce values well within a pixel's width indicates that it can produce very accurate offset values given enough data. This is true both for single and multiple layers offset.

5.2 Design Evaluation

5.2.1 The Alignment module

With the design of the alignment module explained in section 3.4 the high modularity of the pCT-Online package is continued. It is based on proven design patterns that allow for a manageable continuation of development. The implemented software performs within expectations and should, with minor changes, be able to perform simple alignment correction.

This design is optimal for further development on the alignment module, but it also makes it easier to use it in combination with other modules if needed.

The implementation of this module introduced several key concepts of alignment, and it also highlighted several things that need to be researched and implemented in future work.

5.2.2 The Coordinate Transformer

From a performance point of view the current implementation of the coordinate transformer is the most optimal. It is designed around a definition of the geometry of the DTC through a C++ struct. This geometry allows for easy calculations of distances between individual layers, staves, chips, and pixels. The solid throughput of the current implementation should allow for real-time transformation which is optimal for the pCT project.

Some issues with the current implementation has been noted, like the use of basic arithmetic to do transformation, this makes the individual functions somewhat complicated, which a more object based implementation could

fix. Because the transformation task in itself is not too complicated this approach would only increase time and was therefore not used. Testing other ways of implementation has to be done to find the most optimal way of transformation for the pCT project.

5.3 Summary

For this thesis, a new system for the Bergen pCT project was researched and implemented, namely the Alignment module, responsible for detecting misalignments of elements inside the Digital Tracking Calorimeter(DTC). Because this system had little to no previous work done on it before, several parts had to be implemented to get a working prototype of this module. The resulting prototype has acceptable performance suitable for further use and works as a baseline for further research on this topic.

The coordinate transformer implemented to transform local detector data into a global x,y, and z coordinate was made with performance in mind. This is because transforming the data in real-time as it is collected is optimal. Benchmark results indicate that real-time transformation is possible with the current implementation making this a reliable option for further usage.

As a collection, the work done here provides insight into implementations that work and what is needed from future iterations of the Alignment module. It also provides tools needed to perform research on other systems developed for the DTC that relies on global hit information.

Chapter 6

Further Work

6.1 Improving the Alignment Module

To further increase the accuracy and performance of the alignment module, several steps can be done. This refers to all parts of the current alignment module. Some of the most apparent are discussed here.

6.1.1 Per Element Track Filtration

The used method relies on the fact that calculations need many tracks to result in a stable offset value. If an element has too few tracks traversing through it, this will lead to incorrect calculations. Therefore implementing a filter on every element which defines a metric for how certain an offset value is could be done, preventing wrong alignment values.

6.1.2 Regression Models

Several linear regression models used for different types of data exist. Trying a regression model which is more robust to outliers could result in better accuracy. It would also counteract the systematic skew that happens when there is a large offset.

6.1.3 Error Handling

Basic error handling is already present in all parts of the alignment module. A more complex error handling system could be implemented to log errors as they happen. Errors can be grouped into terminating and non-terminating errors, making the system more rigid. Terminating errors will stop the program if they happen, and non-terminating could simply be written to a log file. The log file can be used to detect problems with the system and provide feedback of the processing performance.

6.1.4 Database

A history of the individual elements offsets over time may become helpful in the future, alongside when the most recent alignment was done. It is easily done through the implementation of a database, as mentioned in section 4.3.2.

6.1.5 Advanced Alignment Algorithms

Implementing a more advanced alignment handler capable of making calculations on all alignment parameters must be done. There are already several alignment algorithm examples available for various particle detectors that can be used for inspiration, e.g., the millipede II algorithm[39]. As far as the design of the DTC goes, examples can become more difficult to find.

This task is relatively advanced and could take substantial time to do correctly. Many examples were found during research for this thesis, but all were poorly documented for implementation standards or source code simply inaccessible.

6.2 Evaluating External Tools

6.2.1 Real time Coordinate Transformation

Testing the coordinate transformer on a real-time data stream to see how it affects the system must be done. Most important here is time usage, but measuring the relative resource usage of the system could also be beneficial.

6.2.2 Alternative Coordinate Transformation Methods

Implementing different variations of the coordinate transformer to make it more maintainable can be done. An example of this is an object-based variant "constructing" the DTC through objects and using this to do the transformation. What is essential here is to keep the transformation fast such that it can be used in real-time.

Bibliography

[1] Euclid Seeram. *Computed Tomography - E-Book: Physical Principles, Clinical Applications, and Quality Control*. en. Google-Books-ID: DTCDCgAAQBAJ. Elsevier Health Sciences, Sept. 2015, p. 2. ISBN: 978-0-323-32301-7.

[2] *Docker*. en-US. URL: https://www.docker.com/ (visited on Apr. 20, 2022).

[3] *Valgrind*. URL: https://valgrind.org/docs/manual/cl-manual.html (visited on Apr. 19, 2022).

[4] Marcel van Herk. "Errors and Margins in Radiotherapy." eng. In: *Seminars in Radiation Oncology* 14.1 (2004). Abstract. ISSN: 1053-4296. URL: https://linkinghub.elsevier.com/retrieve/pii/S1053429603000845.

[5] Helge Egil Seime Pettersen. "A Digital Tracking Calorimeter for Proton Computed Tomography." In: *Department of Physics and Technology* (2018), pp. 3–10. URL: https://bora.uib.no/bora-xmlui/handle/1956/17757.

[6] *proton vs photon*. Visited: 09.11.2021. URL: https://commons.wikimedia.org/wiki/File:Dose_Depth_Curves.svg.

[7] *proton vs photon in tissue*. Visited: 09.11.2021. URL: https://commons.wikimedia.org/w/index.php?curid=27983203%20..

[8] *Protonsenter i Bergen*. nn. URL: https://helse-bergen.no/om-oss/protonsenter-i-bergen (visited on May 22, 2022).

[9] Alf Kristoffer Herland. "Development and implementation of data acquisition software for proton computed tomography." eng. In: (June 2021), pp. 1–22. URL: https://bora.uib.no/bora-xmlui/handle/11250/2770399.

[10] *The ALICE detector*. Visited: 10.11.2021. URL: https://home.cern/science/experiments/alice.

[11] Ilker Meric et al. “WP 1 - RADIATION ENVIRONMENT AND ELECTRONICS.” en. In: (2017). URL: https://wiki.uib.no/pct/index.php/File:PCT-WP1-02-Rev3_(Radiation_environment_and_electronics).pdf.

[12] M. Mager. “ALPIDE, the Monolithic Active Pixel Sensor for the ALICE ITS upgrade.” In: 824 (2016), pp. 434–438. ISSN: 0168-9002. DOI: https://doi.org/10.1016/j.nima.2015.09.057.

[13] Johan Alme et al. “A High-Granularity Digital Tracking Calorimeter Optimized for Proton CT.” en. In: *Frontiers in Physics* 8 (Oct. 2020), pp. 1–3. ISSN: 2296-424X. DOI: 10.3389/fphy.2020.568243.

[14] “Software Engineering Course (SWEBOKv3.0).” en-US. In: (2014), p. 169, (9–8). URL: https://www.computer.org/education/bodies-of-knowledge/software-engineering.

[15] Alan M. Christie. “Simulation: An Enabling Technology in Software Engineering.” In: (1999), pp. 25–30. URL: https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=29627.

[16] “Introduction to Monte Carlo simulation.” In: ISSN: 1558-4305. Dec. 2008, pp. 91–100. DOI: 10.1109/WSC.2008.4736059.

[17] *Monte Carlo method*. en. Page Version ID: 1065902265. Jan. 2022. URL: https://en.wikipedia.org/w/index.php?title=Monte_Carlo_method&oldid=1065902265.

[18] Jarle Rambo Sølie. “A Monte Carlo simulation framework for performance evaluation of a proton imaging system without front trackers.” eng. In: (Dec. 2020), pp. 11–12. URL: https://bora.uib.no/bora-xmlui/handle/11250/2716854.

[19] *Ordinary least squares*. en. Page Version ID: 1058372385. Dec. 2021. URL: https://en.wikipedia.org/w/index.php?title=Ordinary_least_squares&oldid=1058372385.

[20] ROOT team. *About ROOT*. en. URL: https://root.cern/about/ (visited on Mar. 11, 2022).

[21] Erich Gamma et al. *Design patterns: elements of reusable object-oriented software*. English. Chapter 1. Reading, Mass.: Addison-Wesley, 1994. ISBN: 0-201-63361-2.

[22] Dhananjay Prasanna. *Dependency Injection: Design patterns using Spring and Guice*. en. Google-Books-ID: PzkzEAAAQBAJ, Chapter 1. Simon and Schuster, July 2009. ISBN: 978-1-63835-301-0.

[23] *Dependency injection*. en. Page Version ID: 1063154604. Jan. 2022. URL: https://en.wikipedia.org/w/index.php?title=Dependency_injection&oldid=1063154604.

[24] *Dependency injection*. URL: https://en.wikipedia.org/wiki/Dependency_injection#/media/File:W3sDesign_Dependency_Injection_Design_Pattern_UML.jpg.

[25] *Modern C++ Design*. en. Page Version ID: 1047387365. Sept. 2021. URL: https://en.wikipedia.org/w/index.php?title=Modern_C%2B%2B_Design&oldid=1047387365.

[26] Andrei Alexandrescu. *Modern C++ design: generic programming and design patterns applied*. en. C++ in-depth series. Chapter 1. Boston, MA: Addison-Wesley, 2001. ISBN: 978-0-201-70431-0.

[27] *Strategy pattern*. URL: https://en.wikipedia.org/wiki/Strategy_pattern#/media/File:Strategy_Pattern_in_UML.png.

[28] "Generic programming." en. In: *Wikipedia* (Nov. 2021). Page Version ID: 1056046095. URL: https://en.wikipedia.org/w/index.php?title=Generic_programming&oldid=1056046095.

[29] David R Musser and Alexander A Stepanov. "Generic Programming." en. In: (1988), p. 19. URL: http://stepanovpapers.com/genprog.pdf.

[30] Øistein Jelmert Skjolddal. "Scalable Readout for Proton CT." eng. In: (Oct. 2020). Publisher: The University of Bergen, pp. 22–23. URL: https://bora.uib.no/bora-xmlui/handle/1956/24109.

[31] H. E. S. Pettersen. "Proton Tracking Algorithm in a Pixel Based Range Telescope for Proton Computed Tomography." eng. In: (2020). DOI: https://doi.org/10.48550/arXiv.2006.09751.

[32] Alba Garcia Santos. "Optimization of the Track Reconstruction Algorithm in a Pixel Based Range Telescope for Proton Computed Tomography." en. In: (2019). URL: https://repository.tudelft.nl/islandora/object/uuid%3Ae8d3f687-43ff-4d5f-a594-2e7d805f146b.

[33] *1. Introduction — GATE documentation*. URL: https://opengate.readthedocs.io/en/latest/introduction.html (visited on Feb. 6, 2022).

[34] J. Allison et al. "Geant4 developments and applications." In: *IEEE Transactions on Nuclear Science* 53.1 (Feb. 2006), Abstract. ISSN: 0018-9499. DOI: 10.1109/TNS.2006.869826.

[35] *Law of large numbers*. en. Page Version ID: 1063288449. Jan. 2022. URL: https://en.wikipedia.org/w/index.php?title=Law_of_large_numbers&oldid=1063288449.

[36] *ECMA-404*. en-US. URL: https://www.ecma-international.org/publications-and-standards/standards/ecma-404/ (visited on Jan. 11, 2022).

[37] *JSON*. URL: https://www.json.org/json-en.html (visited on Jan. 11, 2022).

[38] *Benchmark*. Apr. 2022. URL: https://github.com/google/benchmark (visited on Apr. 19, 2022).

[39] Alice collaboration. "Alignment of the ALICE Inner Tracking System with cosmic-ray tracks." en. In: *Journal of Instrumentation* 5.03 (Mar. 2010). ISSN: 1748-0221. DOI: 10.1088/1748-0221/5/03/P03003.

Appendix A

Source code

The source code for the alignment module is available at this URL: `https://git.app.uib.no/pct-public/pct-online`.

Contributed files:

Under the alignment folder:

1. Everything under /alignment/include/alignment
2. Everything under /alignment/test
3. All loose files under /alignment

   Files under /alignment/include/json is a imported library that can be found here `https://github.com/open-source-parsers/jsoncpp`.

Under the io-adaptors folder:

1. /include/io-adaptors/AlignmentMCSimInputHandler.h
2. /include/io-adaptors/AlignmentToySimInputHandler.h
3. AlignmentMCSimInputHandler.cxx
4. AlignmentToySimInputHandler.cxx

Under the geometry folder:

1. DetectorConstants.h

Under the datamodel folder:

1. /include/datamodel/AliTrack.h

Tools and scripts used are available at this URL: `https://github.com/runalmaas/pCT-Alignment-Tools`. Everything here was created to analyze input and output data from the alignment module and coordinate transformer. Keep in mind that most scripts have outdated formats.

The toy simulation code is available at this URL: `https://git.app.uib.no/pct-public/pru_datasim`.

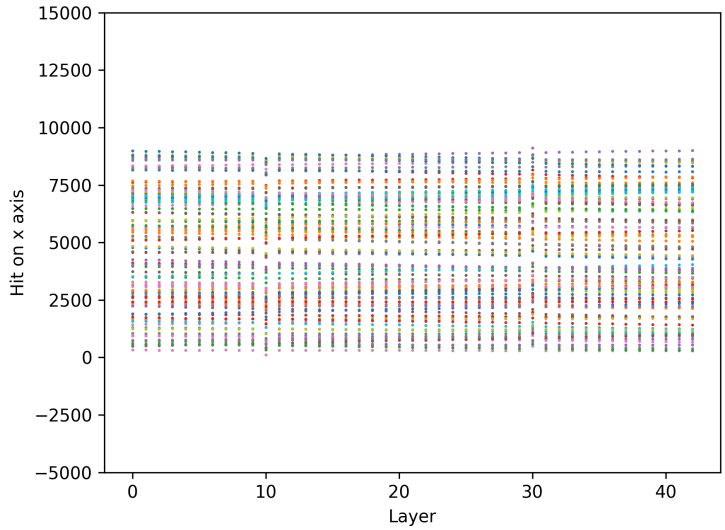
Contributed files can be found under the /simulation folder.

The Monte Carlo simulation code is available at this URL: `https://git.app.uib.no/pct/ztt-monte-carlo-func-test`. This tool was not developed by me.

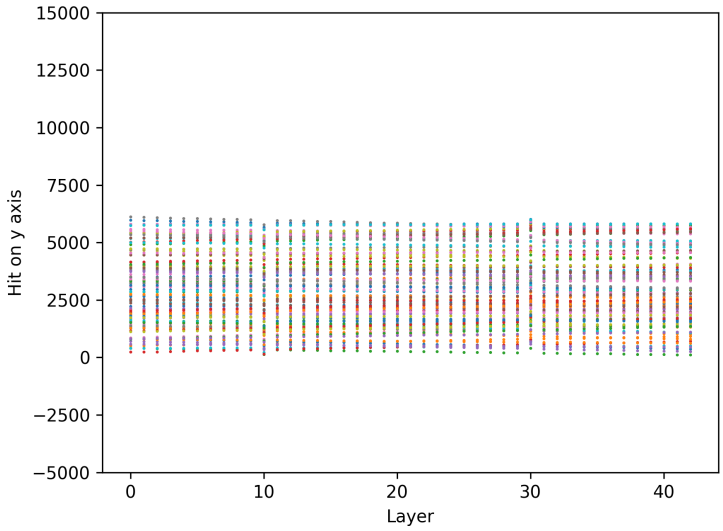
Building the pct-online software is relatively complicated. A guide on how to build all the code should be available within each repository's README.md. A detailed guide is available at this URL: `https://wiki.uib.no/pct/index.php/Building_pct-online_software`.

Appendix B

Offset Correction

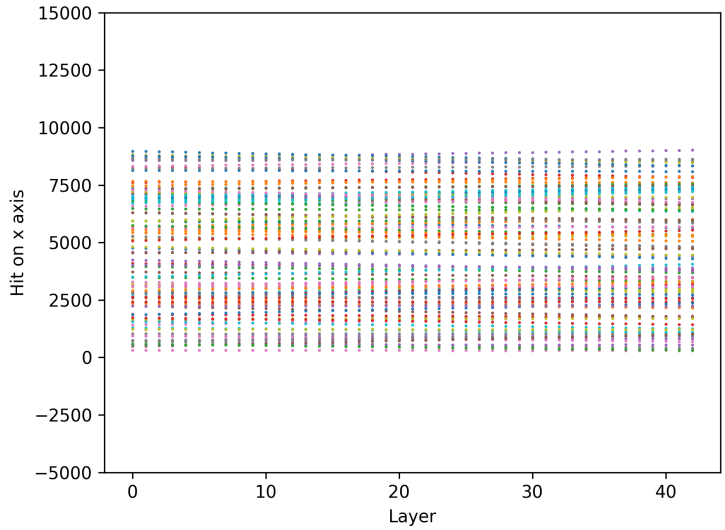


(a)

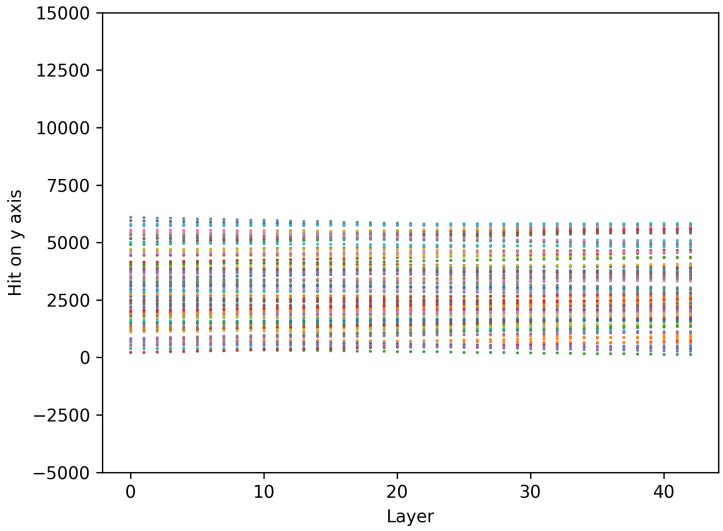


(b)

Figure B.1: *Original 100 tracks with offset on layer 10 and 30 for x-axis(a) and for y-axis(b). Values are given as readout data in a global perspective.*



(a)



(b)

Figure B.2: *The 100 tracks from Figure B.1 with corrected values for x-axis(a) and for y-axis(b). Values are given as readout data in a global perspective.*

Appendix C

Readout chain

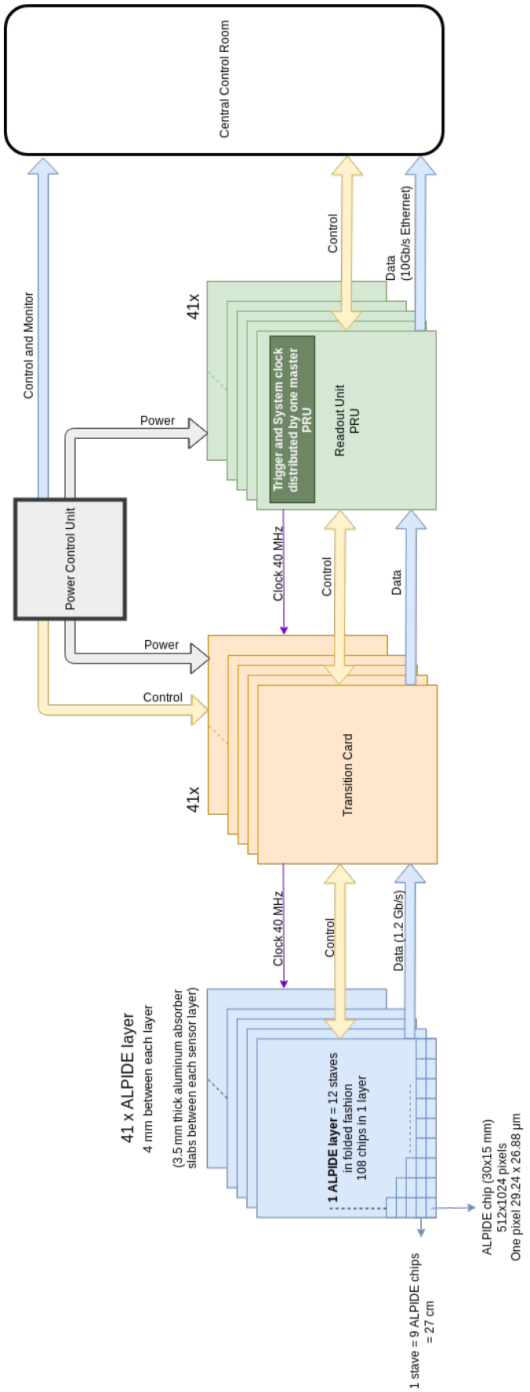


Figure C.1: *The general structure of the proton readout chain[9].*