



**FACULTY  
OF MATHEMATICS  
AND PHYSICS**  
Charles University

**BACHELOR THESIS**

Penda Smajljaj

**Analysing data from two-layer Timepix3  
detector**

Department of Software and Computer Science Education

Supervisor of the bachelor thesis: RNDr. František Mráz, CSc.

Study programme: Computer Science with a  
specialization in Artificial  
Intelligence

Prague 2024

I declare that I carried out this bachelor thesis on my own, and only with the cited sources, literature and other professional sources. I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In ..... date .....

Author's signature

To begin with, I would like to express my deepest appreciation to my thesis supervisor RNDr. František Mráz, CSc. from the Faculty of Mathematics and Physics, Charles University, for his valuable advice and guidance throughout the whole writing and implementation of my thesis. His consistent availability to discuss the progress of my work has been greatly appreciated.

I would like to extend my deepest gratitude to my thesis advisor MSc. Benedikt Ludwig Bergmann, PhD. from the Institute of Experimental and Applied Physics, Czech Technical University, for his helpful suggestions and support.

I am also grateful to my colleagues from the Institute of Experimental and Applied Physics for their insights.

Special thanks should also go to my family for their unwavering support and profound belief in my work. Their encouragement has been instrumental in pushing me toward my goals.

Last but not least, I want to thank my closest friends for their constant encouragement and companionship throughout this journey. I especially want to thank them for making challenging times (often late nights at the library) fun and enjoyable.

Title: Analysing data from two-layer Timepix3 detector

Author: Penda Smajljaj

Department: Department of Software and Computer Science Education

Supervisor: RNDr. František Mráz, CSc., Department of Software and Computer Science Education

Abstract:

Hybrid pixel detectors, such as the two-layer Timepix3 detector, can record huge amounts of information. Compared to the one-layer Timepix3 detector, there are at least twice as many features that can be used for particle analysis or for improving algorithms for partitioning clusters into groups representing a unique particle.

For processing such huge amounts of data, we implemented a tool for cluster visualization, analysis, and filtering. Such a tool can also be used to reevaluate previous groupings of clusters by additionally using spatial information. For convenience, batch processing of files is also included via the command line. Along with batch processing, tasks like filtering out proton clusters, or creating and using regression models on the datasets are also available.

Keywords: two-layer Timepix3 detector, cluster, particle tracking

# Contents

<b>Introduction</b>	<b>7</b>
<b>1 Timepix Detectors</b>	<b>9</b>
1.1 Introduction to Elementary Particle Detection . . . . .	9
1.2 Hybrid Pixel Detectors . . . . .	9
1.3 The Medipix Collaboration . . . . .	10
1.4 Timepix3 . . . . .	11
1.5 Timepix4 . . . . .	12
1.6 Two-layer detectors . . . . .	12
1.7 Clustering . . . . .	12
<b>2 Data Format and Features Description</b>	<b>15</b>
2.1 Using ROOT . . . . .	15
2.2 Cluster and File Design – Definitions . . . . .	15
2.3 Identifying Coincidence Groups . . . . .	17
<b>3 Related Work</b>	<b>20</b>
<b>4 Goals</b>	<b>21</b>
4.1 Purpose . . . . .	21
4.2 Usage . . . . .	22
<b>5 Analyser</b>	<b>23</b>
5.1 Used Technologies . . . . .	23
5.2 Features . . . . .	23
5.3 Analysis of Events . . . . .	23
5.3.1 Global Analysis . . . . .	24
5.3.2 Local Analysis . . . . .	24
5.4 Filtering . . . . .	25
5.4.1 Filtering of Clusters . . . . .	25
5.4.2 Filtering of Coincidence Groups . . . . .	25
5.5 Coincidence Group Reevaluation . . . . .	25
5.5.1 Selecting the Right Particles for Reevaluation . . . . .	26
5.5.2 Two Main Ideas for Checking Proximity . . . . .	27
5.5.3 Applying Constraints . . . . .	30
5.5.4 Implementation . . . . .	31
5.6 Estimating $\theta$ . . . . .	32
5.7 Batch Processing Through The Command Line . . . . .	33
5.7.1 Filtering Out Protons . . . . .	34
5.7.2 Customizable Multivariate Linear and Quadratic Regressor	34
5.7.3 Using Pre-trained Models for Predictions . . . . .	34
5.8 Results and Visual Output . . . . .	35
5.8.1 Local Analysis . . . . .	35
5.8.2 Global Analysis . . . . .	35
5.8.3 Filtering . . . . .	42

5.8.4	Estimating $\theta$ . . . . .	43
5.8.5	Coincidence Group Reevaluation . . . . .	43
5.8.6	Filtering Out Protons . . . . .	48
5.9	Runtime analysis . . . . .	49
5.10	Drawbacks . . . . .	52
<b>6</b>	<b>Incorporating Machine Learning Techniques</b>	<b>53</b>
6.1	Machine Learning . . . . .	53
6.2	Program Objectives with Machine Learning . . . . .	54
6.3	Techniques to Use and Why . . . . .	54
6.3.1	Linear Regression . . . . .	54
6.3.2	Quadratic Regression . . . . .	57
6.4	Implementation . . . . .	58
6.4.1	Estimating $\phi$ . . . . .	58
6.4.2	Setting Parameters of the Reevaluation Ellipse Based on $\theta$ . . . . .	58
6.4.3	Customizable Linear and Quadratic regressor . . . . .	60
6.5	Results . . . . .	61
6.5.1	Estimating the Azimuth Angle $\phi$ . . . . .	61
6.5.2	The Effect on Estimations of the Azimuth and Other Parameters on Coincidence Group Reevaluation . . . . .	61
6.5.3	Using Pre-trained Models for Predictions . . . . .	64
	<b>Conclusion</b>	<b>65</b>
	<b>Bibliography</b>	<b>66</b>
	<b>List of Figures</b>	<b>69</b>
	<b>List of Tables</b>	<b>71</b>
	<b>Appendices</b>	<b>72</b>
<b>A</b>	<b>User Documentation</b>	<b>72</b>
A.1	Dependencies . . . . .	72
A.2	Installation . . . . .	72
A.3	Running Analyser . . . . .	74
A.3.1	GUI usage . . . . .	74
A.3.2	Command Line Usage . . . . .	84
A.3.3	Filtering Protons Out . . . . .	88
A.3.4	Training Regression Models . . . . .	88
A.3.5	Using a Pre-trained Model for Predictions . . . . .	88
<b>B</b>	<b>Attachments</b>	<b>90</b>
B.1	Analyser - Source Code . . . . .	90
B.2	Analyser - Executables . . . . .	90
B.3	Analyser - User Documentation . . . . .	90
B.4	Link to the Test Data . . . . .	90
B.5	Link to GitLab Repository . . . . .	90

# Introduction

Physicists study matter starting from the fundamental building blocks of the universe – elementary particles. They study how these particles interact with each other and how they tend to combine to form something bigger. However, these particles are tiny, so physicists depend on various instruments to detect them. A result of such endeavors is the invention of so-called hybrid pixel detectors.

These detectors rely on segmented sensors, which, in addition to the energy deposition in the material, provide the topology of events and have been mainly used for tracking particles close to the interaction point and reconstructing tracks of particles passing through the detector. With the help of computer scientists, a lot of information from such a history of events can be inferred. This is very useful for more complex tasks, such as determining the type of particles you are observing and so on.

In my thesis, I will work with a two-layer Timepix3 detector setup, where coincident tracks enable to capture more information about the registered particles. In contrast to Timepix1 and Timepix2 detectors, Timepix3 has more advanced features, allowing us to work with the data we get more efficiently. However, the amount of data that these detectors produce is huge. So, we need to find a way to process and infer new information from it without requiring too much time, and for this reason, we need to be careful with what technologies we use to achieve that.

## Thesis Outline

This thesis aims to build a tool for analyzing data from the two-layer Timepix3 detectors. This analysis will include many features on its own, and there will also be improvements to some methods already used on such datasets, like the reevaluation of the labeling of clusters into coincident groups or estimating some attributes of an event in these datasets differently from what was already done. This will allow for recognizing any irregularities that occur in the datasets more easily, visually representing the events, and using the datasets to get new ones by filtering or reevaluating.

The rest of the thesis is divided into 9 more chapters:

- Chapter 1 gives an introduction to elementary particle detectors and the clustering of particles.
- Chapter 2 explains how the datasets we get from the detector are structured.
- Chapter 3 talks about related work that has been done in this area of study.
- Chapter 4 talks about the purpose and usage of my application.
- Chapter 5 describes the implementation and the features included.
- Chapter 6 describes the incorporation of machine learning into the application and how it affects the results.
- The conclusion describes the results that were achieved during the whole project.

- Appendix A contains the user documentation for the application developed within the thesis.
- Appendix B describes the contents of the electronic attachment of the thesis and gives a link to the GitLab repository with the source code of the developed application and a link to sample datasets.



# 1 Timepix Detectors

In this chapter, I will talk about the origin and development of particle detectors over the years. I present notable particle detectors that were predecessors to Timepix3, the detector used to obtain the datasets I use in this thesis. Then, I will introduce Timepix3 in more detail, describing its important properties. I will close this chapter by discussing the clustering process and introducing the main types of clusters we typically see in the data.

## 1.1 Introduction to Elementary Particle Detection

As described in [1], the evolution of particle detectors originated when Henri Becquerel discovered radioactivity in 1896, and Wilhelm Conrad Röntgen made a huge discovery on the identification of X-rays shortly thereafter. The earliest nuclear particle detectors, such as X-ray films and primitive zinc-sulfide scintillators, were very simple yet crucial in studying processes like  $\alpha$ -particle scattering. Shortly after Sir William Crookes' accidental observation of light flashes from radium salt on activated zinc sulfide in 1903 led to the creation of the spintariscope, a particle detector still in use today for demonstrations [2]. [3] Cherenkov particle detectors, first unknowingly observed by Madame Curie in 1919, also made huge progress in particle physics [4].

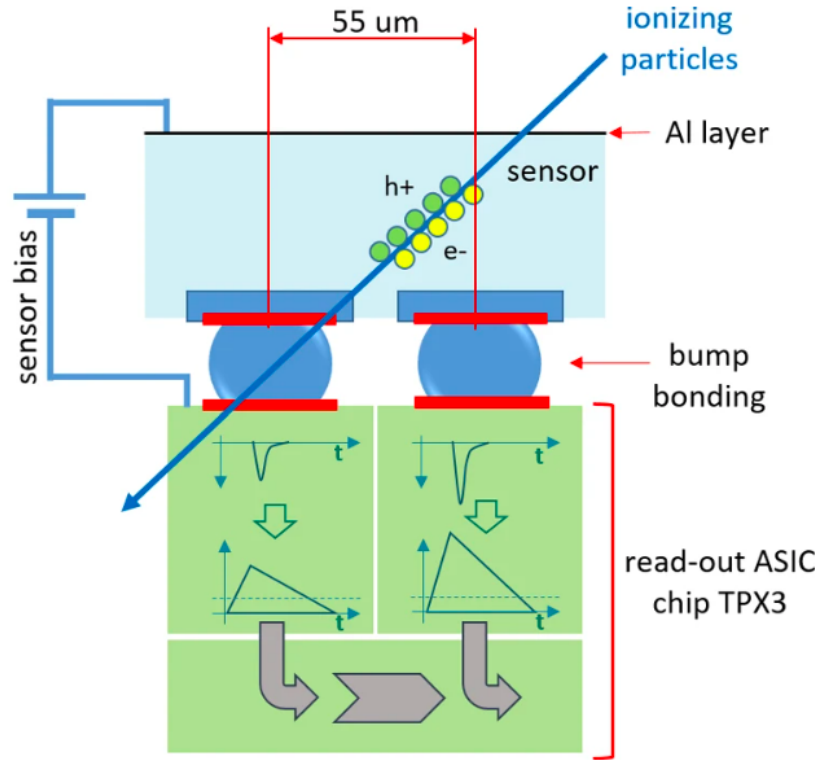
However, scientists today do not just want to detect elementary particles. They want to be able to get more information about the particle, like what particle it is, how much energy and momentum it possesses, what the spatial coordinates of the particle were in order to build a trajectory of the particle, and so on. For this, newer and more practical detectors were developed. Spark chambers, and bubble and cloud chambers use optical detection, and drift chambers and silicon pixel detectors use electronic detection.

Initially, particle detectors were mainly used in cosmic rays and nuclear and particle physics. Over time, they have also become very popular and useful in the fields of medicine, biology, archaeology, arts, and many more. Particle detectors have played a crucial role in advancing science. Innovative detection methods, such as cloud chambers, bubble chambers, and others, have led to important discoveries. Several Nobel Prizes, including those awarded to individuals like C.T.R. Wilson and P. Cherenkov, have recognized the impact of these advancements in the field.

## 1.2 Hybrid Pixel Detectors

Hybrid pixel detectors are detectors that originated as a solution to the problem of distinguishing particles that hit the linear detectors that were being used at the time (1990s) simultaneously. Physicists and engineers proposed having pixelated detectors where each pixel has its own electric chain, which can help determine whether it has been "hit" by a particle. Such a detector was first described in 1988 [5]. This is how hybrid pixel detectors were born.

These detectors have two main parts: an integrated circuit (read-out chip) and a pixelated (semiconductor) sensor. The sensor is usually silicon, GaAs (Gallium arsenide), or CdTe (Cadmium telluride), and its main job is to detect the incident radiation (ionizing particles) in the form of an electron cloud or hole. On the other hand, the read-out chip contains an electronic chain for every pixel of the sensor. The surface of this electronic chain is the same as that of the sensor pixel – about a hundred  $\mu\text{m}$  square. To connect this electronic part with the sensor, they used a technique called bump-bonding (see Figure 1.1).



**Figure 1.1** The active layer of a Timepix3 detector bump-bonded to the chip of the Timepix3 detector where the signal is processed (source [6])

Since hybrid pixel detectors were essentially designed to detect ionizing radiation, they are often also called ionizing pixel detectors. They are mainly used in High Energy Physics (HEP) experiments, space missions, and in X-ray imaging and CT scans [7].

### 1.3 The Medipix Collaboration

Improving medical diagnosis with minimal patient side effects requires an imaging system that offers clear pictures, efficient detection, quick results, and low radiation. X-ray digital radiography, introduced in the 1980s [8], has evolved with technologies like CCD readout and amorphous silicon detectors [9]. To address efficiency and contrast issues, direct X-ray conversion through semiconductor detectors, like the “photon counting chip” (later called Medipix), was developed, allowing for better image quality.

In 1997, a collaboration between CERN (Conseil Européen pour la Recherche Nucléaire) and a few universities in Freiburg, Glasgow, and Italy was created. The first prototype (Medipix-1) had a screen of 64x64 170  $\mu\text{m}$  pixels. The new property that was introduced in Medipix detectors was a 15-bit pixel counter that would keep a record of the individual particles that make contact with it, and an energy threshold [10].

After the success of Medipix-1, they started working on Medipix-2, a chip of 256x256 pixels of 55  $\mu\text{m}$ . Medipix-2 was the first hybrid pixel detector with a pixel pitch below 170  $\mu\text{m}$ . It was mainly designed for radiography applications. Now, as an addition, there were two energy thresholds and a 13-bit counter, which counts and records the number of individual photons or particles that interact with the detector. One of the core features and benefits of Medipix-2 was the high spatial resolution it provided, which is very important for medical and X-ray imaging. This also became the most important attribute of the Medipix family as a whole.

Based on the Medipix-2 detector, a new type of detector was designed. By introducing a clock for each pixel, they now had the possibility of counting the number of clock ticks when the energy detected at the pixel was above a particular energy threshold (from when a particle is detected to when it goes away). Now, two new attributes were added: Time-over-Threshold (ToT) and Time-of-Arrival (ToA). Time-of-Arrival is calculated as the time from when a trigger goes off to when it detects radiation amount with energy above the specified threshold. This detector was then called “Timepix”.

The latest Medipix chip – Medipix-3 – introduced “charge summing mode” and the ability to have numerous thresholds with individual counters each, which makes it possible to have continuous readout (see [7]). The charge summing mode was a solution to double counting, which was a notable problem for previous Medipix detectors as it would so happen that a unique photon would be accounted for by two neighboring pixels instead of just the proper one. The energy resolution was significantly improved as a result [11].

## 1.4 Timepix3

Timepix has found many applications with the newly implemented timing system. It was frame-based and used a continuous data-driven mode, where each pixel sends data individually. However, it still had an issue. If multiple particles hit a certain pixel in a very short period of time, then the pulses would accumulate for the Time-over-Threshold mode. That would basically make it impossible to differentiate between different particles since the energy recorded would just sum up. This issue was addressed by its successor – Timepix3.

The detector I will be focusing on in this thesis is Timepix3. As opposed to Timepix, which reads the pixel matrix as a whole, Timepix3 reads the information about a particular particle right after the hit detection, along with the coordinates of the pixel that was hit. As a result, instead of having a continuous stream of large frames, we now have a continuous stream of smaller pieces of data. This event-based continuous read-out system notably reduced the energy accumulation issue in Timepix [10]. Timepix and Timepix3 are also 256x256 pixels, where each pixel is a 55  $\mu\text{m}$  square. Each separate hit of radiation is processed by the

detector through the pixel counters so the detector can also be thought of as 65 535 separate counting detectors.

## 1.5 Timepix4

Timepix4 is the latest detector of the Timepix family. It has 448x512 pixels, and through bump bonding, it is connected to a sensor of pixels of  $55\mu\text{m}$  square. Similarly to Timepix3, it records ToT and ToA information in data-driven mode every time a particle registers energy higher than the defined thresholds of the detector but with improved time resolution. Timepix4 also has photon counting mode. The Timepix4 ASIC (Application-Specific Integrated Circuit) is possibly the first large area, high-resolution pixel detector readout ASIC that can be tiled on all 4 sides. This is important if we want to cover large surfaces smoothly so that we do not implement rooftop-like structures [12].

## 1.6 Two-layer detectors

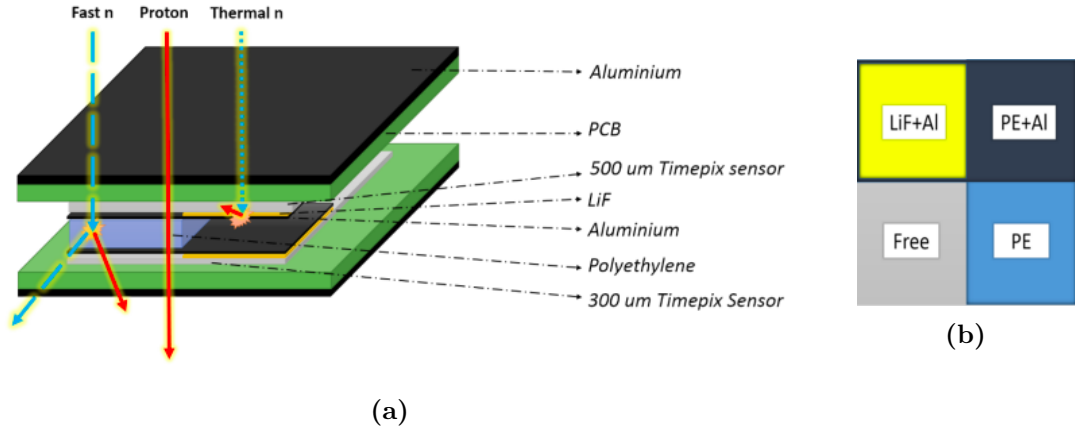
Two-layer detectors, also known as detectors with the ATLAS-TPX design, are made of two Timepix detectors. This design has two sensors with a thickness of  $500\mu\text{m}$  each [13]. These two layers are positioned to face each other. To detect or distinguish different types of particles, we can put a layer of material between the two layers. This is especially useful for recognizing neutrons as this layer is made of thermal and fast neutron converters. This design takes advantage of the fact that neutrons have no electrical charge; they may not be detected by the first layer, but before they reach the second one, they will be converted or decomposed into charged particles when they hit the converter layers, which makes it easier for us to detect them. After detection, we can tell them apart from  $\gamma$ s by looking at the cluster type (explained in the next section) and from charged particles by using time coincidence information.

These neutron converters define four separate regions as shown in Figure 1.2. The region behind the 1.2 mm thick polyethylene (PE) enables directional detection of recoil protons from fast neutron interactions; the PE+Al region reduces the number of low-energy recoil protons; the third region, covered by LiF foil, facilitates thermal neutron detection through  $\alpha$ -particles and tritons; and the uncovered Si region is designated for background subtraction.

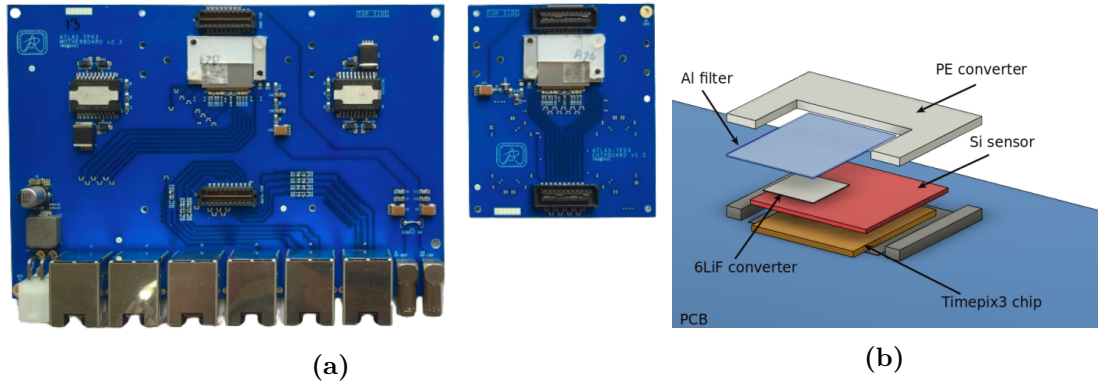
The benefits of this design (see Figure 1.3) are the possibility to track charged particles and the observation of coincidences and anticoincidences, which is particularly useful for distinguishing between neutrons and energetic ions [14].

## 1.7 Clustering

When we are getting data from Timepix3, we will see that a certain particle hitting the detector will produce a substantial number of lit-up pixels depending on the energy it has and the path it takes. A cluster is formed by spatially and temporally coinciding hits, representing a unified grouping of detection events. In the original Timepix, however, only spatial information is used, which poses

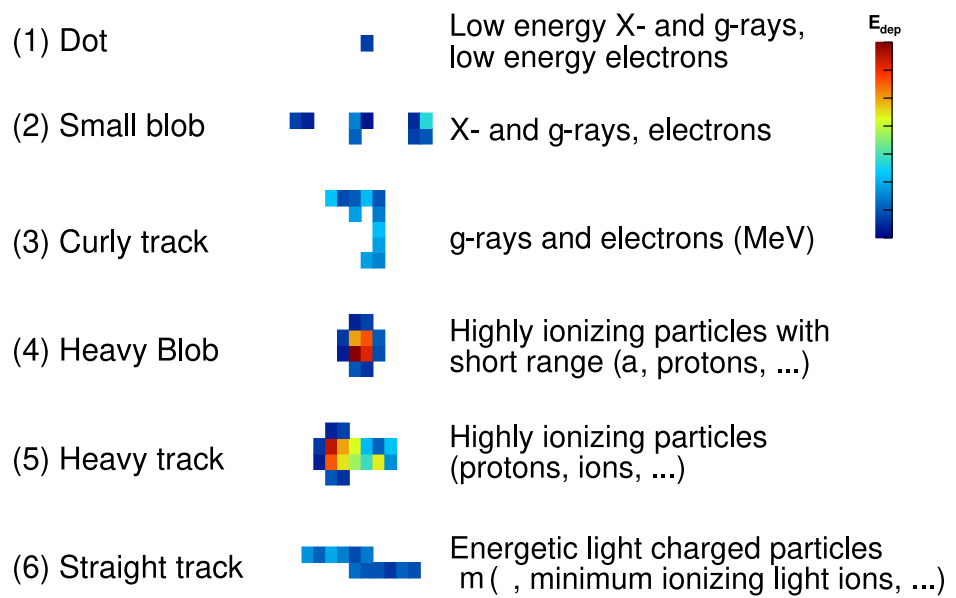


**Figure 1.2** Composition of the two-layer detector (Reproduced from [14] under CC 4.0.). (a) Charged particle tracking and neutron detection (b) Layout of the converters



**Figure 1.3** Design of the two-layer detector (Reproduced from [13] under CC 4.0.). (a) Side view of the fully assembled device; (b) Front view of the fully assembled design

a problem in the case of clusters overlapping in space since we cannot classify whether they are the same or not without taking the time they first appeared into consideration. The types of clusters, as described in [14], are: dots (one-pixel clusters), small blobs (round clusters with at most four pixels), heavy blobs (round clusters with at least five pixels), heavy tracks (elliptical or straight clusters with at least three pixels), straight tracks (long straight clusters with at least 20 pixels in length and less than three pixels in width), and curly tracks (everything else), although more complex types of clusters can be observed that have not been strictly classified yet. You can see it visually described in Figure 1.4. When it comes to neutrons, they will often be found as heavy blobs and heavy tracks since they are usually detected through low energy protons,  $\alpha$ -particles, and tritons. That is how the clusters are formed, and they form the basis of the datasets I will use in this thesis.



**Figure 1.4** Cluster types classification (Reproduced from [15] under CC 4.0.)

# 2 Data Format and Features Description

In high-energy physics, the ROOT framework (see [16]) is a de facto standard for storing data on elementary particles. Therefore, we will describe the framework below, and our developed tools will assume input as a ROOT file and produce some output in this format. In the following sections, I will explain the usage of the ROOT framework and cluster design, and I will end by introducing the concept of coincidence groups.

## 2.1 Using ROOT

ROOT, developed at CERN, is an open-source framework for data processing primarily used in high-energy physics. It is mainly used to analyze data and perform simulations. ROOT provides tools for efficient storage (compressed binary data), manipulation, and analysis of large datasets. ROOT files thus contain data structures and objects used for storing and retrieving complex data, often in the form of experimental measurements and simulations.

The ROOT files that I will be using represent measurements taken in well-defined particle beams at the Danish Center for Particle Therapy in Aarhus [17] (protons from 80-240 MeV) and the Los Alamos Neutron Science Center (neutrons from 1-600 MeV) [18]. The setup for these measurements is seen in Figure 2.1. These measurements show different types of particles hitting the two-layer TimePix3 detector. Using histograms, we can display them graphically and extract information from them that will help us arrive at various conclusions about these particles as a whole. The particles are examined in terms of clusters, where each cluster is a group of pixels from the detector closely tied together, which get “lit up” when the particle goes through them.

## 2.2 Cluster and File Design – Definitions

Each of the ROOT files I used is organised in a TTree with many branches that represent the attributes of a cluster, such as:

- **cluster size** – the amount of pixels that get lit up when a particle passes through the detector,
- **pixels in the X axis** – the list of  $x$ -coordinates of the pixels that get lit up,
- **pixels in the Y axis** – the list of  $y$ -coordinates of the pixels that get lit up,
- **cluster mean X** – the center of the position of the cluster in terms of the  $x$  direction,
- **cluster mean Y** – the center of the position of the cluster in terms of the  $y$  direction,

- **cluster volume centroid  $\mathbf{X}$**  – the center of the position of the cluster in the  $x$ -axis based on its energy,
- **cluster volume centroid  $\mathbf{Y}$**  – the center of the position of the cluster in the  $y$ -axis based on its energy,
- **times of arrival (ToA)** – the list of exact moments in time when a certain pixel detected the particle, and this may differ very slightly between different pixels of the cluster,
- **times over threshold (ToT)** – the list of time intervals in which the voltage output stays over a threshold for each pixel of the cluster,
- **min ToA** – the smallest time of arrival recorded on a pixel of the cluster,
- **delta ToA** – the difference in time between the first and last recorded ToA of the cluster,
- **coincidence group** – an ID (integer starting from 0) that identifies a group of clusters close together in time assumed to be part of the same particle,
- **coincidence group size** – the number of clusters in the same coincidence group,
- **layer** – the layer of the detector the cluster was detected on,
- **cluster volume** – the energy of the cluster in keV,
- **cluster height** – the maximum energy in keV of a single pixel of the cluster,
- **phi** – the azimuthal angle of the projection of the particle in the  $xy$ -plane,
- **theta** – the polar angle of the particle and the  $z$ -axis in three-dimensional space, which also represents the deviation angle from the normal to the surface of the detector,
- **linearity** – a value between 0 and 1 that represents how linear a cluster is, where 0 means the cluster is not linear, and higher values mean more linearity. It is defined as

$$\frac{N_{\text{inline}}}{N}, \quad (2.1)$$

where  $N_{\text{inline}}$  (inline pixel count) is the longest sequence of pixels in one line and  $N$  represents the total number of pixels,

- **roundness** – a value between 0 and 1 that represents how round a cluster is, where 0 means the cluster is not round, and higher values mean more roundness. It is defined as

$$\frac{2\sqrt{\pi N}}{l_{\text{border}}}, \quad (2.2)$$

where  $l_{\text{border}}$  (border length) is the number of pixels on the border and  $N$  represents the total number of pixels,



- **diameter** – defined as

$$\frac{2\sqrt{N}}{\pi}, \quad (2.3)$$

where  $N$  represents the total number of pixels,

- **cluster type** – indicates what cluster type this cluster belongs to, according to the classification in Figure 1.4,
- **inner pixel count** – counts the number of pixels that are not neighboring with any pixel that does not belong to the same cluster,
- **border pixel count** – the number of pixels on the border (opposite from inner pixels),
- **cluster length** – only usable for tracks with linearity above 90% and refers to the distance between the two pixels furthest away from each other; it is defined as

$$\sqrt{(d_{\max} \times d_{\text{pitch}})^2 + t_{\text{sensor}}^2}, \quad (2.4)$$

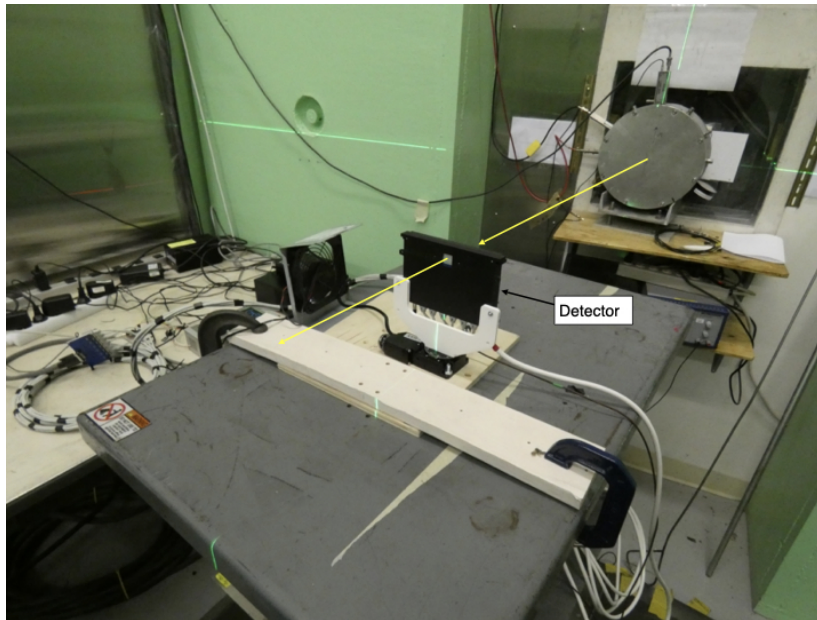
where  $t_{\text{sensor}}$  represents detector thickness,  $d_{\text{pitch}}$  represents detector pixel pitch, and  $d_{\max}$  represents maximal distance,

- **edge** – true if the cluster touches the border of the pixel screen, otherwise false,
- **dx** – the (max) difference along the  $x$  axis,
- **dy** – the (max) difference along the  $y$  axis.

## 2.3 Identifying Coincidence Groups

When a particle goes through the detector, more than one cluster may be generated in the output as a result. It is important that we try to identify which of the clusters observed are likely to belong to the same particle so that we can attempt to distinguish between different particles. Such a set of clusters that are believed to represent one specific particle is called a *coincidence group*. So far, the usual way of assigning clusters to a coincidence group was by setting a very small time frame and partitioning the event space into time intervals of length equal to that time frame, so that any two events (clusters) whose time of arrival information differs by less than the specified time frame are considered to be part of the same coincidence group. We can look at examples of such coincidence groups in Figure 2.2.

So, when we assign coincidence groups to clusters, we only take the time of arrival into account. However, this is not the most accurate way to approach this. It may often happen that we see different particles overlapping in time (they appear in the same time frame), and by the method explained above, all of the resulting clusters will get assigned to the same coincidence group, which should not be the case because they do not all represent the same particle. One way to improve this method is to also consider the spatial information, which I will explain in the later chapters, together with all the features of my program.



Experimental setup of the fast neutron measurement performed at the Los Alamos Neutron Science Center in a white neutron beam with neutron energies from 1-600 MeV. The neutron beam direction is indicated by the yellow arrows. The two-layer Timepix3 detector is labelled.

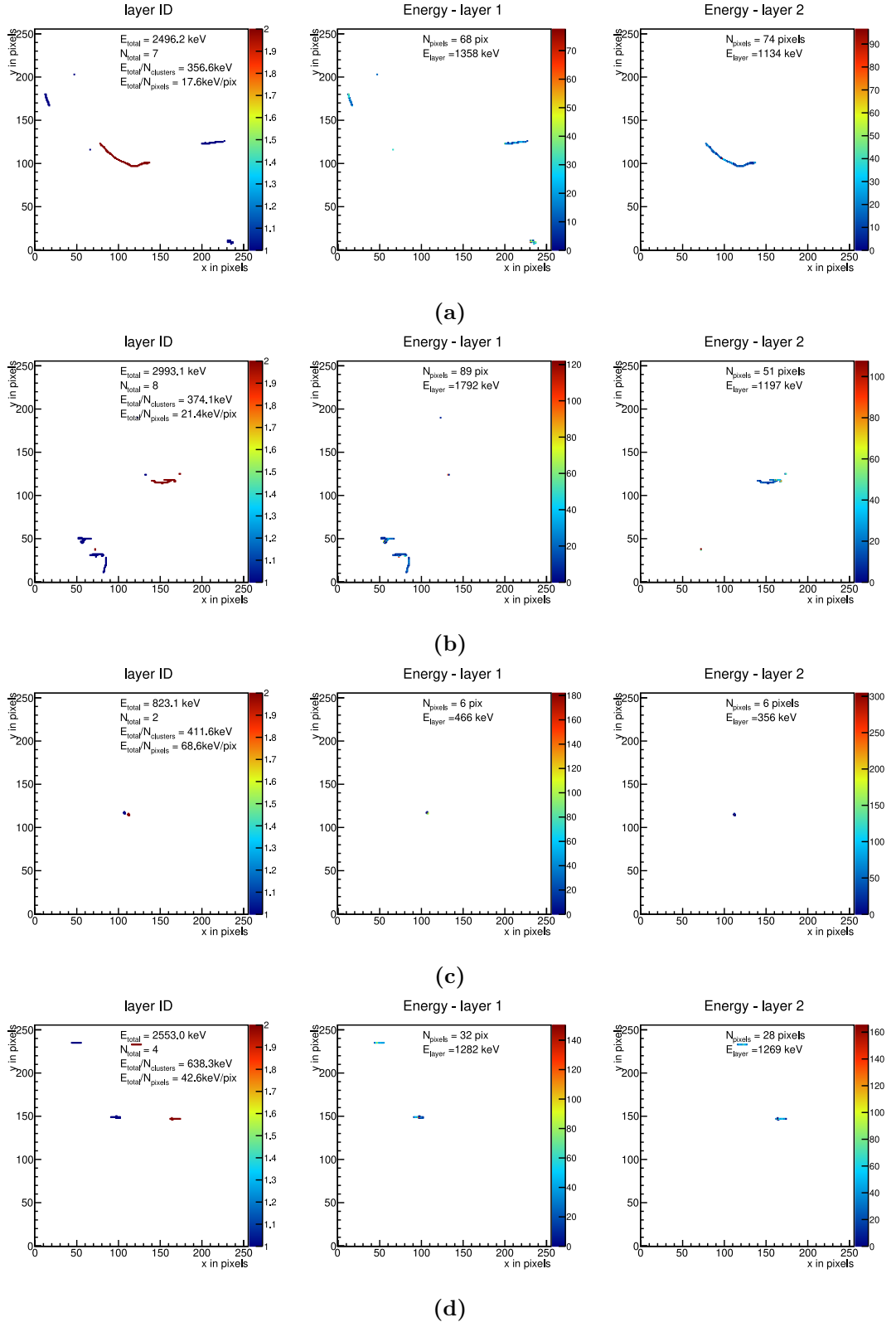
(a)



Experimental setup of the proton measurements. The device was irradiated with protons of different energy at the Danish Center for Particle Therapy. The proton beam direction is indicated.

(b)

**Figure 2.1** Experimental setup for the proton and fast neutron measurements.



**Figure 2.2** (a) and (b) represent two different coincidence groups for neutrons detected in a fast neutron beam where the long tracks observed are likely electrons or photons; (c) coincidence group for a proton approaching the detector at a perpendicular angle; (d) coincidence group for a proton approaching the detector at an angle of  $50^\circ$  away from the normal, where we can observe that two different particles appear as one coincidence group.

## 3 Related Work

The ROOT datasets I use in my thesis represent the sequence of events recorded by the two-layer Timepix3 detector in terms of clusters. This is very beneficial for me because I did not need to do clustering myself. Multiple people worked on that before me and implemented several clustering techniques. Such techniques are mentioned in theses and papers from Lukáš Meduna and Tomáš Čelko. Lukáš Meduna proposed a clustering framework [19]. This framework could be used to visualize and classify clusters, perform real-time clustering algorithms, and for data acquisition. In his master's thesis [20], Tomáš Čelko proposes various clustering algorithms, some of which implement parallelization for faster processing, some implement approximation clustering where, for runtime purposes, there is a little less focus on the quality of clustering, and some offer partial clustering based on requirements defined by the user.

Many people are working with the Timepix3 detector, so similar work was done before for analyzing data. However, a big difference is the fact that most of the work is done on one-layer Timepix3 detectors instead of two-layer ones. For this reason, more research can be done with data from the two-layer Timepix3 detectors, where my thesis can be useful.

Notable papers to mention as related work are papers by Tomáš Čelko, Petr Mánek, and P. Smolyanskiy. Lukáš Meduna also worked on detecting particles on the Timepix3 detector [21]. He implemented a tool for viewing, analyzing, and classifying particles for one-layer Timepix3 detectors. In his bachelor's thesis [22], Tomáš Čelko implemented a tool for cluster visualization and filtering, which overlaps with what I do in my thesis for two-layer Timepix3 detectors. He also used machine learning to classify clusters, a feature that could be implemented next as an extension of my thesis.

Regarding the two-layer Timepix3 detector, the design was introduced in a paper [13] by P. Smolyanskiy et al. It was implemented as an attempt to distinguish charged particles from neutrons by using coincidences of clusters in different layers and to allow more accurate computations of the impact angles, which I take advantage of when estimating angles  $\phi$  and  $\theta$  in Sections 5.6 and 6.4.1.

Another important software that must be mentioned is Track Lab ([23] and [24]), developed by Petr Mánek. Track Lab is a free and open-source application that is used for data acquisition and analysis and has high performance. It is extensible and used with different types of detectors and configurations. However, Track Lab does not include the acquisition and analysis of data from two-layer detectors. Compared to the one-layer Timepix3 detector, the two-layer Timepix3 detector at least doubles the amount of features that can be used for particle classification. For this reason, I will be performing a similar analysis but for two-layer Timepix3 detectors.

# 4 Goals

The two-layer Timepix3 detectors capture more information about the stream of particles than their predecessors. Therefore, being able to analyze and work with such data is very beneficial to physicists in high-energy particle physics. The goal is to build an application as a tool that enables users to do that. The application will be called Analyser. This chapter will give a quick introduction to the features of my application. In Section 4.1, I will talk about the purpose of this application and introduce the goals (functionalities) that the application will fulfill. In Section 4.2, I will briefly describe how the application can be used.

## 4.1 Purpose

The main purpose of my application is to be a tool for analyzing data from the two-layer Timepix3 detector. This application will include many functionalities useful for high-energy particle physicists working with such data. Users can do local and global analyses of events in ROOT files based on coincidence groups. As a result, they get histograms, general statistics about the coincidence groups present, and a visual representation of the events in the file.

Another functionality is filtering clusters and coincidence groups based on criteria specified by the user. Both for analysis and filtering, users can browse through the coincidence groups interactively in the application. Then, there is coincidence group reevaluation by using spatial information, attempting to partition the clusters into coincidence groups better. This feature will improve the current labeling of coincidence groups where only temporal information is considered. These features are all available in the application interface.

However, when the users want to do batch processing and include multiple ROOT files as datasets, the application can also run through the command line. The user can perform global analysis and cluster filtering through the command line. Additionally, three more functionalities are included: filtering out protons, training the provided datasets for a regression model, and using a pre-trained regression model to make predictions. Filtering out protons attempts to remove clusters that likely belong to a proton from a dataset. This is useful for cleaning the dataset and working with what remains to perform other tasks, such as predicting which clusters were produced by neutrons. Training the provided datasets for a regression model is a functionality that allows the user to learn relationships between different attributes of the data using linear or quadratic regression. It can be useful for simple machine learning tasks and can later be extended into more complex techniques. Such a trained model can then be serialized and saved into a file to be used later for testing its performance and making predictions on certain datasets.

Since the application is built using C++, it is efficient to be used with large files, which is very common for the datasets provided. It will still generally take some minutes for a whole file to be processed. However, it is still faster than using other programming languages.

## 4.2 Usage

The application will have a Graphical User Interface (GUI), making it easy for users to see all the tools they can use to analyze, filter, or reevaluate coincidence groups in a particular file. Another way to use the application is through the command line. This is better when the user is performing batch processing of files. Based on the parameters specified by the user, he/she can perform global analysis, cluster filtering, filtering out protons, training datasets on a regression model, and using pre-trained regression models for predictions.

# 5 Analyser

Analyser is a tool that takes ROOT files (datasets) from the two-layer Timepix3 detector and has many features that the user can use to analyze, filter, or modify these datasets. In Section 5.1, I will talk about used technologies. In Section 5.2, I will introduce the features of my application, such as analysis, filtering, and coincidence group reevaluation of events. In Section 5.3, I will explain the analysis of events. In Section 5.4, I will talk about filtering clusters and coincidence groups. In Section 5.5, I will explain the implementation of coincidence group reevaluation and how it works. In Section 5.6, I will show how to estimate the angle  $\theta$  using information from both layers. In Section 5.7, I will talk about batch processing of ROOT datasets. In Section 5.8, I will show the results and visual output of my application. In Section 5.9, I will analyze my application's performance in different tasks and input files. In Section 5.10, I will conclude this chapter by noting some drawbacks of the application and what could be improved.

## 5.1 Used Technologies

My goal was to build a program with a Graphical User Interface (GUI) that would use ROOT files as described above and process them for further analysis. ROOT files recording clusters detected with the two-layer Timepix3 detectors are usually gigabytes in size. To work with such large files, we need an efficient code. Therefore, e.g., Python is not suitable, and I will be using C++ to process such files since it is way more efficient and faster than other programming languages, and it traditionally works well with ROOT files. For the Graphical User Interface, I will use Qt ([25]) as it is a popular choice for C++ applications and fits well with my project.

## 5.2 Features

The main features of this program will be a global and local analysis of events, filtering of events, and coincidence group reevaluation for data from two-layer Timepix3 detectors. Some of the analyses (mostly when it comes to processing the whole file) can take a long time to run. It can also happen that the user wants to work with more than one input ROOT file at once. Therefore, some of the functionality, including additional features like batch processing of files, training regression models, predicting datasets using pre-trained models, and filtering out protons, will be possible to use from a command line. These features will be implemented in an interactive GUI application or using a command line interface. A detailed description of the features will be given in subsequent sections.

## 5.3 Analysis of Events

This program will be a tool for global and local analysis of events (clusters) on the two-layer detectors. In this section, we will distinguish between analysis

types. Section 5.3.1 describes the global analysis, while Section 5.3.2 describes the local one.

### 5.3.1 Global Analysis

The global analysis will be an analysis of a whole ROOT file. It will give some general characteristics of the file, like the number of coincidence groups and some statistics about the types of coincidence groups encountered in the dataset.

The statistics will include information like the percentage of coincidence groups with unequal cluster count between layers, the percentage of coincidence groups that only appear on the second layer, the percentage of coincidence groups that have more than 1 cluster in layer one, and the percentage of coincidence groups containing clusters of some minimal size (minimal size to be determined by the user).

Global analysis will also include some histograms showing cluster count and cluster size in layers 1 and 2 of the detector. Additional histograms include histograms about the recorded values of the angles  $\phi$  and  $\theta$  and their estimations, whose computation is explained in detail in Sections 5.6 and 6.4.1. They can be useful for seeing the distribution of such angles and deciding whether the distribution of the estimated values shows us something the original distribution does not. This is also helpful for seeing how pure our dataset is when it comes to the distribution of angles  $\phi$  or  $\theta$ . Lastly, four more histograms are included, each showing the cluster energy recorded in a particular region of the detector, defined by the converters between layers 1 and 2.

During interactive viewing, along with browsing through individual coincidence groups, the user can see estimations of the value of  $\theta$  for coincidence groups with one-to-one matches of clusters between layers. This can be helpful for recognizing outliers or can be valuable information for evaluating the dataset.

### 5.3.2 Local Analysis

Local analysis of a cluster will provide the user with some general characteristics about it. At first, the user selects a cluster she is interested in, and then she will obtain some information about the cluster, which can include the size, the pixels it lights up, the time of arrival for each pixel, the time over threshold for each pixel, the coincidence group it belongs to, the detector layer it appeared on, the cluster volume in keV (which denotes the “energy”), the  $\phi$  and  $\theta$  angles, the cluster type and possibly more. Coincidence groups detect different clusters in both layers, and the difference in properties of the clusters in different layers can be valuable information for the user.

Local analysis can also be used for coincidence groups. The user will select what coincidence she wants to inspect by specifying its ID (an integer starting from 0 that represents that coincidence group). Then, the user can see a visual representation of that coincidence group, including some information about it, such as the total number of clusters, the number of clusters in each layer, and its total energy in keV.



## 5.4 Filtering

Filtering is producing a set of results after removing the examples we do not want to see. There will be two types of filtering: by clusters and by coincidence groups. They both have their own special filters the user can apply. There is also the option of applying multiple filters at once but only within the category (so not mixing cluster filters with coincidence group filters).

As an output, viewing the filtered results interactively or in batch mode will be possible. By “interactively” I mean that the user will be able to view the filtered clusters one by one, by clicking “next” or “previous”. In batch mode, the filtered result will be displayed in a PDF file, where the maximum size of the PDF can also be limited if the user does not want to deal with having a very big PDF file. As an additional feature, the filtering result will also be saved as a new ROOT file containing only the filtered clusters’ data.

Section 5.4.1 describes the filtering of clusters in more detail, while section 5.4.2 talks about the filtering of coincidence groups.

### 5.4.1 Filtering of Clusters

There are different types of cluster filters the user can apply. He/she can choose to see only clusters of a certain size, or of a certain length or type (only blobs or tracks, for example), clusters that appear for only a certain period of time, or clusters that come from a specific angle. Even additional attributes of the dataset can be used to filter clusters. The features not included as an option can be added by using a cut. A cut is a string representing the condition to be applied to the data in the tree. It can combine multiple conditions using logical operators like `&&` (logical AND), `||` (logical OR), and `!` (logical NOT). You can use parenthesis and ROOT built-in mathematical functions (such as `abs()`, `sin()`, `sqrt()`, and so on), and the logical operators follow the usual precedence. After the user filters them out, the resulting ROOT and PDF files will contain only coincidence groups that do not contain such clusters.

### 5.4.2 Filtering of Coincidence Groups

When it comes to filtering coincidence groups, the user can filter them based on whether they have multiple clusters in one layer overlapping (approximately) in time, have unequal cluster numbers between layers, have only clusters in layer two, or have clusters of size bigger than the minimal cluster size (the user can specify what size is minimal before she filters). After she filters them out, the resulting PDF will show only the remaining coincidence groups.

## 5.5 Coincidence Group Reevaluation

When I explained how coincidence groups are assigned, I mentioned that that was not the most accurate grouping of clusters because it was based only on the time of arrival and it often happened that we had cases where different particles overlapping in time got assigned to the same coincidence group. I will try to make this process a bit more accurate by analyzing such coincidence

groups with temporally overlapping clusters once more and generating one or more coincidence groups out of it, which will be not solely based on time of arrival but on proximity as well. In Section 5.5.1, I will talk about selecting the right particles for reevaluation. In Section 5.5.2, I will introduce the two main ways of doing proximity checks. In Section 5.5.3, I will talk about additional constraints for removing bad clusters from consideration. In Section 5.5.4, I will describe the implementation in code of the reevaluation methods.

### 5.5.1 Selecting the Right Particles for Reevaluation

If we look at one cluster in layer 1 and want to decide what clusters from layer 2 can be considered to be of the same particle, we must expect the particle to behave a certain way. To be able to take proximity into account as well when we are assigning coincidence groups to clusters, we should have some idea regarding the patterns that a particle typically follows as it goes through the detector. For neutrons, for example, if we see some cluster on layer 1, it is very hard to determine where to look for in layer 2 to identify clusters that could belong to or be a result of the same neutron. This is because when a neutron travels between the two layers of the detector, it creates secondary particles in layer 2 that can be scattered in any direction. In general, its behavior is very unpredictable. When it comes to protons, however, predicting where to look in layer 2 to find corresponding clusters is very intuitive.

When going through the detector, a proton typically produces one cluster in layer 1, and then at a certain distance, it produces another one in layer 2. The distance between the two clusters and the shape or length of the cluster it produces depends on both angles  $\theta$  and  $\phi$ . However, in the data sets that have been measured so far, the particles are mostly parallel to the  $x$ -axis, and then the problem reduces to a 1-D task, where the patterns of proton clusters we observe are mostly dependent on the angle  $\theta$  at which they hit the detector.

When  $\theta$  is zero, the direction of the incoming proton is normal to the detector, the clusters it produces are in the form of small blobs, and the two clusters between layers will appear only a few pixels in difference. On the other hand, when  $\theta$  is 90, and the protons are close to parallel to the surface of the detector, they will appear as very long straight tracks and mostly appear in one layer only. Sometimes, these long tracks will span most of the  $x$  axis.

Other possible values for  $\theta$  from the proton datasets I have observed so far are 30°, 50°, 120°, and 180°. Due to the rotational symmetry of the device design, the behavior of protons when  $\theta$  is 180° is analogous to that of protons where  $\theta$  is 0°. There is also very close alignment in the behavior of protons when  $\theta$  is equal to 50° and when it is equal to 120°. These similarities make sense when you consider the fact that the pair of  $\theta$  angles 0° with 180° has the same deviation from the normal. The pair of  $\theta$  angles 50° with 120° differ by only 10 degrees in the deviation from the normal, so the results they produce are quite similar, too.

When talking about pairs of angles that produce analogous behavior, it is common to wonder whether we can see differences in time-of-arrival of the pixels in layers 1 and 2 between two datasets of different angles and use this for further analysis. Intuitively you would expect that for  $0^\circ \leq \theta < 90^\circ$ , since particles are coming to the detector from the front, the time-of-arrival of pixels recorded in

layer 1 is lower than that of pixels recorded in layer 2, and the opposite true for  $90^\circ \leq \theta \leq 180^\circ$ . However, the detector's time resolution of 2 ns is not sufficient to resolve the particle direction of flight. Hence, no matter what value  $\theta$  has, this information on time-of-arrival for each pixel does not really show us anything useful for analysis.

It should be noted that as the deviation from the normal grows, the clusters that a proton produces go from small blobs to straight tracks whose length grows along with the deviation from the normal. Another change is the expected distance from the cluster it produces in layer 1 to the one it produces in layer 2 (calculated as the distance between the centers of energy for each cluster), which also grows along with the deviation.

Hence, since the pattern of protons is easier to predict, I will perform coincidence group reevaluation only for protons for now (excluding those of  $\theta = 90^\circ$  since the clusters are very long and usually only in one layer), although I should note the fact that after more thorough studying of the behavior of other particles, the idea can be extended to be used for other particles, too.

### 5.5.2 Two Main Ideas for Checking Proximity

Considering the patterns of clusters that protons produce that we discussed in the previous section, we should do coincidence group reevaluation in a way that fits with what we intuitively expect to see and adjust the evaluation appropriately according to  $\theta$ . Now, going to the idea behind the implementation, we should mention the fact that the coincidence groups are already assigned based on time of arrival information, so I will process these already existing coincidence groups and additionally add a proximity check to reevaluate them. In the following subsections, I will explain the two main ideas to use for checking proximity: neighborhoods and reevaluation ellipses.

#### Using Neighbourhoods

Say we select an already existing coincidence group, and we assume that it has  $C = \{C_1, \dots, C_m\}$  clusters in layer 1 with  $|C| \geq 1$  and  $\bar{C} = \{\bar{C}_1, \dots, \bar{C}_n\}$  clusters in layer 2, where  $n = |\bar{C}|$  is the total number of clusters in layer 2. We select one cluster of layer 1; let's denote it as  $C_x$ . Let  $G_x$  denote the coincidence group, which includes  $C_x$  and all clusters in layer 2 that are at an appropriate distance from  $C_x$ . I should also note that to define the  $x$  and  $y$  coordinates of a cluster, I will consider the attributes defined in the dataset: *clstrVolCentroidX* and *clstrVolCentroidY*. They define the center of energy of a cluster.

Now, we want to determine which clusters of  $\bar{C}$  are close enough in distance to  $C_x$  to conclude that they were a result of the same particle and should therefore be assigned to the same coincidence group. The simplest way to check the proximity would be to use an  $\varepsilon$ -neighborhood of  $C_x$  and some distance function  $d$  between two clusters, such that  $\forall \bar{C}_y \in \bar{C} : d(C_x, \bar{C}_y) < \varepsilon \implies \bar{C}_y \in G_x$ .

The simplest way to imagine this is by selecting the point  $(C_x.clstrVolCentroidX, C_x.clstrVolCentroidY)$ , which denotes the center of energy of the cluster  $C_x$  in the pixel plane, and then drawing a circle around it whose radius is  $\varepsilon$ . Then, we can see which cluster from layer 2 lays inside this circle (this will also be evaluated only by looking at the coordinates of the center

of energy of the cluster) and assign it to  $G_x$ . As we can see in an example of this in Figure 5.2b, this works well for protons with  $\theta = 0^\circ$  or  $\theta = 180^\circ$ , if we decide on an appropriately small  $\varepsilon$ . Experimentally, I saw that  $\varepsilon = 15$  pixels was a good enough parameter, so I have been working with that for now. Nonetheless, we should note that this method only works for these types of protons because they appear as small blobs and generate clusters in very close proximity to each other.

## Introducing Reevaluation Ellipses

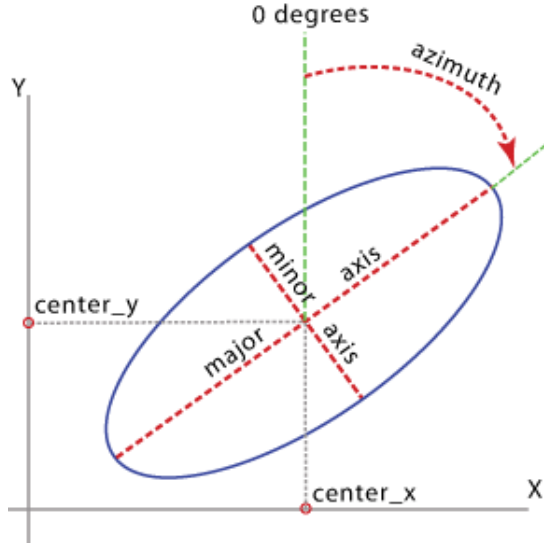
As we described earlier, the patterns of protons change as the deviation from the normal of the detector plane changes. For this reason, we cannot count on a neighborhood as much to help us group clusters correctly and must think of another way. However, there is a very simple extension of the initial idea that we can adjust according to  $\theta$ , and it would do the same job. Instead of thinking of a circle around the point  $(C_x.clstrVolCentroidX, C_x.clstrVolCentroidY)$ , we can now think of a reevaluation ellipse with the same center. The reevaluation ellipse (shown in Figure 5.1) will have three important parameters that need to be set: the major, the minor, and the rotation (azimuth) angle  $\phi$ .

Luckily,  $\phi$  will correspond to the angle of the initial cluster  $C_x$  that we are centering the reevaluation ellipse on. So,  $\phi = C_x.phi$ . This was easy to ignore for protons of  $\theta \in \{0^\circ, 180^\circ\}$ . Since they go through the detector perpendicularly and only leave a small blob,  $\phi$  becomes quite useless and is not even determined in the dataset. In the other types of protons, though (specifically talking about protons with  $\theta \in \{30^\circ, 50^\circ, 120^\circ\}$ ), this feature becomes very useful. If we just have an ellipse with no sense of direction, it becomes much harder to group clusters correctly. This is because we should expect that if it is the same proton that produced two clusters, the second one will appear in the same direction as the first one, hence we must rotate the reevaluation ellipse the same way in order to capture this relationship.

However, it is safer not to always rely on the fact that  $\phi$  is given or set correctly in all datasets. We should have some way of determining it on our own given just the  $xy$ -coordinates of the cluster, and the user can then choose this option when reevaluating the coincidence groups for less dependence on the accuracy of  $\phi$  given in the dataset. To calculate it ourselves, we use the “line of best fit” idea (fitting a line through the  $x$  and  $y$  data points to determine the angle  $\phi$ ), which I will explain in detail in Section 6.4.1, and ignore it for clusters with *roundness*  $> 0.8$ , in which case we use neighborhoods and  $\phi$  becomes useless.

Another option is for the user to set the value of  $\phi$  directly, and it will then be the same for all coincidence groups. This can generally work if the user knows that a certain value of  $\phi$  is common for a very high percentage of clusters, in which case it will not impact the result negatively too much. Such analysis of  $\phi$  values throughout the dataset can be done during global analysis, as described in Section 5.3.1.

The next thing we have to figure out is how to set the major and the minor axes. The major  $a$  will determine the width of the reevaluation ellipse; meanwhile, the minor  $b$  will determine the height. Estimating the minor and the major axes of the reevaluation ellipse should also account for measurement errors as we deal with experimental data. Since protons tend to follow quite a linear path, we do not need the minor to be that high. Typically, I set  $b$  between 20 and 25. As for



**Figure 5.1** Visual representation of a reevaluation ellipse where we see how the azimuth ( $\phi$ ) is defined, along with the major and the minor axes (image source [26])

the major, this depends more on the value of  $\theta$ . The further  $\theta$  is from the normal, the bigger the major should be to enable considering bigger distances between two clusters (as discussed earlier, distance can be bigger the bigger the deviation is). For protons of  $\theta \in \{50^\circ, 120^\circ\}$ , since their behavior is similar, I experimentally decided to set  $a$  to 120, and for protons of  $\theta = 30^\circ$ , to set it to 55.

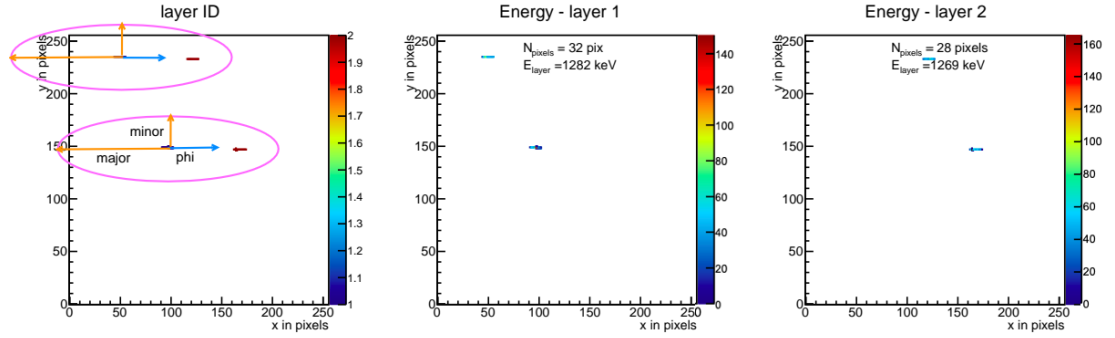
However, it is essentially necessary to figure out a function or a relationship between  $\theta$  and the major and minor axes of the reevaluation ellipse. We need to know how the major and minor change as the value of  $\theta$  changes and to learn this relationship so that we can use it to determine how to set the parameters of the reevaluation ellipse based on a given value of  $\theta$  without experimentally playing with it ourselves. Here is when quadratic regression comes in handy, and I will explain in more detail how I solved it in Section 6.4.2.

Following this idea, when we have again the cluster  $C_x$  of layer 1, and we are checking whether clusters  $\bar{C} = \{\bar{C}_1, \dots, \bar{C}_n\}$  of layer 2 can be assigned to the same coincidence group as  $C_x$  ( $G_x$ ), we say that

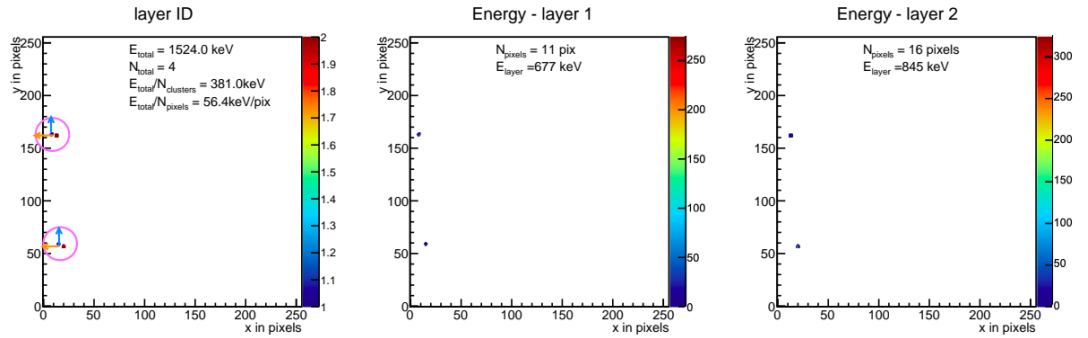
$$\forall z \in \bar{C} : z \in G_x \iff \frac{((z_x - x_x) \cos \phi + (z_y - x_y) \sin \phi)^2}{a^2} + \frac{((z_x - x_x) \sin \phi - (z_y - x_y) \cos \phi)^2}{b^2} \leq 1 \quad (5.1)$$

where  $z_x$  and  $z_y$  correspond to the coordinates of the center of the cluster  $z$  defined as  $(z.clstrVolCentroidX, z.clstrVolCentroidY)$ ,  $x_x$  and  $x_y$  correspond to the coordinates of the center  $C_x$  (which is also the center of the reevaluation ellipse) defined as  $(x.clstrVolCentroidX, x.clstrVolCentroidY)$ ,  $\phi$  is the rotation angle (azimuth) of the reevaluation ellipse,  $a$  is the major of the reevaluation ellipse, and lastly  $b$  is the minor. This is essentially what we use to determine whether to consider a cluster  $z \in \bar{C}$  to be a result of the same proton that produced  $C_x$  or not. We can look at some examples of this working in Figure 5.2. We can also notice that when the major equals the minor, what we get is a circle (major = minor =  $\varepsilon$ ), and we go back to the first idea of using a neighborhood for protons

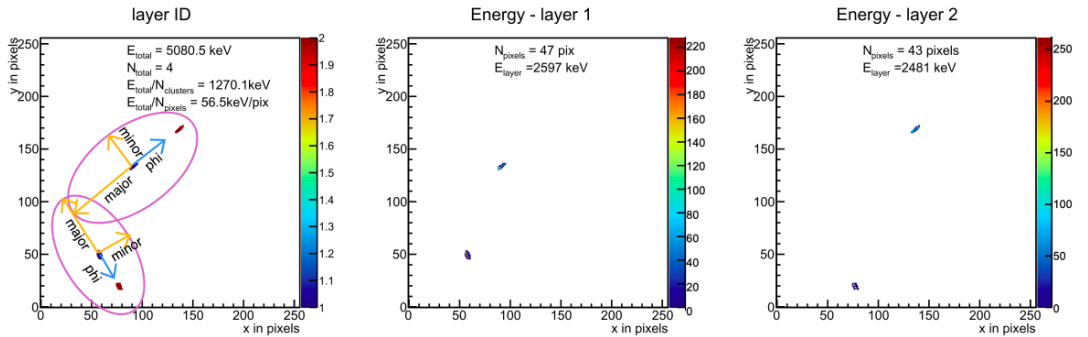
of  $\theta \in \{0^\circ, 180^\circ\}$ .



(a) Reevaluation of protons of  $\theta$  value 50 using a reevaluation ellipse where we see two different coincidence groups that were originally considered to be in the same coincidence group be reevaluated based on proximity.



(b) Reevaluation of protons perpendicular to the surface of the detector where we use a neighborhood to test proximity and separate two different coincidence groups.



(c) Reevaluation of protons of  $\theta$  value 30, where we see two different protons with different directions  $\phi$ , and see how the reevaluation ellipse rotates to capture such a relationship between clusters correctly.

**Figure 5.2** How sample reevaluation ellipses look like for different types of protons.

### 5.5.3 Applying Constraints

Sometimes, only checking for candidate clusters from layer 2 using a neighborhood or an ellipse may not be enough. It may happen that two clusters are in close proximity to each other, but they differ in certain attributes, so when inspecting the visual representation manually, you might say they do not belong to the same particle. However, since we will be processing huge amounts of data, in order to tell the computer what attributes to look for to try and distinguish whether two

clusters are of the same particle or not, we need to select some more important attributes that can help us decide. Such example attributes can be cluster length, cluster roundness, the azimuth angle  $\phi$  of a cluster, and the Euclidean distance between the two clusters.

When it comes to cluster length, we check that the difference in length between the two clusters is less than  $170 \mu\text{m}$ . For cluster roundness, we check that the difference between the two clusters is less than 0.25. We check for both cluster length and cluster roundness because it gives us a more accurate description of the cluster's shape. Since we expect both clusters to be of the same shape, checking both attributes helps us be more exact.

For the azimuth  $\phi$ , we check for the difference to be less than 0.3 radians (17.2 degrees). As for the Euclidean distance between the two clusters, we check for a difference of less than 20 pixels compared to an estimated distance  $d_{xy}$  that can be calculated using the original angle  $\theta$  from the first cluster. This estimated distance is computed using the formula

$$d_{xy} = \tan \theta \Delta z, \quad (5.2)$$

where  $\Delta z$  represents the depth between the two layers of the detector.

The reason we apply this constraint on distance on top of checking proximity through a reevaluation ellipse is so that we can rule out clusters that are too close compared to what is expected (usually useful for bigger values of  $\theta$  for which the distance between the two corresponding clusters between the layers increases significantly so we need to remove candidate clusters in layer 2 that appear in very close proximity). Another thing I have to note is the relatively large difference boundaries for the constraints. This is because it allows the program to be robust and prepared for slight errors that can be induced by the recorded parameters in the datasets used.

#### 5.5.4 Implementation

When implementing this idea in code, going through the ROOT file and reevaluating all of the coincidence groups one by one is not the best idea because not all need to be reevaluated, and it would not be efficient. The coincidence groups that need reevaluating the most are those with time-overlapping clusters in layer 1 since it cannot typically be the case that a simple proton hits the detector at two different places at once in the same layer. However, in order to make the algorithm more robust, I will reevaluate all the coincidence groups so that I can also distinguish outliers (for example, clusters in layer 2 that appear alone and far from all others). So, in order to be as efficient as possible, we will first loop through each coincidence group once to see what clusters we have to consider, and then, we loop through it one more time in order to reevaluate it using the methods explained in the preceding paragraphs.

When we have a coincidence group with time-overlapping clusters, we start with the assumption that every cluster on layer 1 is a coincidence group on its own. Let us denote such clusters as  $C = \{C_1, \dots, C_m\}$ , where  $m$  is the total number of clusters in layer 1, and let  $G_i$  be the new coincidence group that corresponds to cluster  $C_i$ . Then, for each cluster  $C_i \in C$ , we center a reevaluation ellipse at the center of the cluster  $(C_i.clstrVolCentroidX, C_i.clstrVolCentroidY)$  and use the clusters found in layer 2 to determine which of those satisfy:

1. The center of the cluster is inside the reevaluation ellipse (equation 5.1).
2. The difference between  $d_{xy}$  (equation 5.2) and the distance from the center of the cluster to the center of  $C_i$  is at most 20 pixels.
3. The difference in cluster length between the given cluster and  $C_i$  is at most 170  $\mu\text{m}$ .
4. The difference in cluster roundness between the given cluster and  $C_i$  is at most 0.25.
5. The difference between the azimuth  $\phi$  of the given cluster and the azimuth  $\phi$  of  $C_i$  is at most 17 degrees (precisely 0.3 radians).

Clusters specifying all the aforementioned criteria are then assigned to coincidence group  $G_i$ . This is essentially how new coincidence groups are created out of old ones, where proximity and additional constraints are now also considered.

## 5.6 Estimating $\theta$

The angle  $\theta$ , defined as the deviation angle from the normal to the surface of the detector, is very important for analysis, coincidence group reevaluation, and for understanding important properties of the patterns we see on the pixel screen. Although it might not be as significant for neutrons, for example, it actually has significant impact on the categorization of the proton signature. Because of this, we must try to get as accurate estimations of it as possible.

In the datasets I have worked with so far, it is calculated independently for every cluster. The good thing about this is that you can get an estimation for every cluster regardless of whether it has a corresponding cluster in the other layer or not. The formula used to calculate  $\theta$  in degrees in the given datasets for a single cluster is:

$$\theta = \left( \arctan \frac{dp}{t} \right) \frac{180}{\pi}, \quad (5.3)$$

where  $d$  represents the maximal distance between any 2 pixels of the cluster computed as the Euclidean distance between the coordinates of the pixels,  $p$  represents the detector pixel pitch, and  $t$  represents the detector thickness.

While this formula gives us some heuristic for  $\theta$ , it is better to have some perception of the depth or  $z$ -axis included in the formula since  $\theta$  itself represents the polar angle of the particle in three-dimensional space. However, to have a perception of depth, we need each cluster in layer 1 to have a corresponding cluster in layer 2. For this reason, we can improve the estimation of  $\theta$  for coincidence groups where there is a one-to-one match of clusters between the two layers. Coincidence group reevaluation is good for generating such coincidence groups, so they constitute a large number of coincidence groups of datasets.

An important fact to know is that the coordinate system of layer 1 is at  $(0,0,0)$ , and the two layers are not directly under each other, but there is a slight offset in pixels. From the configuration, we know that for protons, in layer 2, the offset given in pixels is

$$\vec{o}_{1 \rightarrow 2} = (\Delta_x, \Delta_y, \Delta_z) = (-5.3, 1.4, -47.4). \quad (5.4)$$



So now whenever we have clusters  $C_x \in C$  and  $\bar{C}_y \in \bar{C}$ , we define

$$(x_x, x_y) = (C_x.clstrVolCentroidX, C_x.clstrVolCentroidY),$$

and now we add the layer 2 *offset* to the coordinates of  $\bar{C}_y$ :

$$\begin{aligned} y_x &= \bar{C}_y.clstrVolCentroidX + \Delta_x, \\ y_y &= \bar{C}_y.clstrVolCentroidY + \Delta_y. \end{aligned}$$

Now we calculate the changes in position as:

$$\begin{aligned} \Delta x &= y_x - x_x, \\ \Delta y &= y_y - x_y, \\ \Delta z &= \Delta_z. \end{aligned}$$

$\Delta z$  now represents the depth between the two clusters. We calculate the hypotenuse in  $xy$  plane (the distance the particle has traveled in the plane parallel to the layers (ignoring the  $z$ -axis)) as

$$d_{xy} = \sqrt{\Delta x^2 + \Delta y^2}. \quad (5.5)$$

Now  $\theta$  is calculated like:

$$\theta = \arctan \frac{d_{xy}}{\Delta z}. \quad (5.6)$$

This estimation of  $\theta$  can be seen when viewing coincidence groups interactively during global analysis. The results (shown by histograms in Figure 5.12) align well with the real  $\theta$  of the datasets tested.

## 5.7 Batch Processing Through The Command Line

Through the graphical user interface (GUI), the user can select one dataset to work with and then decide what he/she wants to do with it. However, sometimes, it is more convenient to work with more than one dataset at once. This could be for efficiency reasons or for convenience.

Using a graphical user interface to process huge datasets, such as the ones I am working with (produced by Timepix3), will slow down the process a little. If the dataset is relatively small, it is not noticeable; however, when just going through the whole file takes a significant amount of time, efficiency becomes a problem. Because of this, using the command line to analyze your datasets can result in faster processing.

The user may also be faced with the problem of having to use the same functionality on multiple datasets. Since you can only process one file at once on the GUI, the user would have to manually wait for each file to be processed and then run the next one. In situations like these, the command line can be of great help. The user can select all the files that need processing and then choose what to do with them.

The functionalities included in the command line are global analysis, filtering of clusters, and three additional ones that I will describe in the subsections below.

In Section 5.7.1, I will talk about filtering out protons. In Section 5.7.2, I will introduce customizable linear and quadratic regressors. Lastly, in Section 5.7.3, I will discuss using pre-trained regression models to make predictions and test the performance.

### 5.7.1 Filtering Out Protons

A task that physicists working with particle detectors have been dealing with for some time now is the possibility of recognizing which clusters in the datasets were produced by neutrons. This task is quite hard due to the nature of detecting neutrons. Neutrons typically produce other particles when traveling between the layers of the detector, and these particles may be scattered in all directions. Hence, instead of dealing with this challenging problem directly, we can reduce it to a smaller one.

One thing we can do to reduce the problem is try to take the protons out of the file with some degree of confidence and then work with what remains to see which ones could be neutrons. Since, as explained in Section 5.5, we already have a way to produce new coincidence groups for protons by taking spatial information into account, we can now use that to filter out one-to-one matches of proton clusters between the layers. This functionality then makes it easier for what will remain in the dataset to be further analyzed to see what could be produced by a neutron and what not.

### 5.7.2 Customizable Multivariate Linear and Quadratic Regressor

Another functionality of the command line is the customizable multivariate linear and quadratic regressor. As explained in detail in Section 6.4.3, this tool will be useful for learning relationships between different attributes of the dataset. On the command line, the user will specify the attributes he/she wants to test, the output attribute, and the datasets on which the model will be trained. Since it is used through the command line, the user can select multiple datasets for training instead of just one. There is also the possibility of choosing between a linear or quadratic regression model depending on the user's requirements.

The main use of this tool at the moment is to analyze how different attributes or features of the dataset are connected. However, as this tool is very simple, it can be easily extended into more complex models for a wider range of applications. One particular use case that can later be developed is a classifier for particle types, which can be a simple extension of the regression models already developed.

After training, the model will be serialized and saved into a file with the extension ".bat". The user can later use this model to make predictions and test its performance.

### 5.7.3 Using Pre-trained Models for Predictions

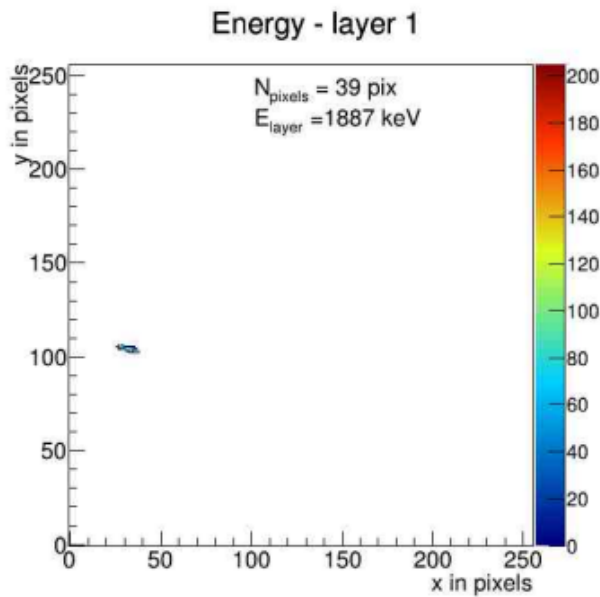
When the user has an already trained regression model, whether it be linear or quadratic, she can use that later to make predictions on datasets of her choice. After predictions are made, she will get a report of its performance by looking at

the Root Mean Squared Error. This functionality is mostly to test the performance of the model and try to find any dependence relations between different features of the data. In the future, this feature could also be useful when we extend such regression models to be used for particle classification. The predictions will be saved into a CSV file that she can inspect and work with later. This CSV file will have only three columns: cluster index, real output value, and predicted output value.

## 5.8 Results and Visual Output

In this section, I will explain and provide many examples of the functionalities of the GUI of my application. In the following subsections, you will read about local and global analysis (Section 5.8.1 and 5.8.2), filtering of clusters (Section 5.8.3), estimating  $\theta$  (Section 5.8.4), coincidence group reevaluation (Section 5.8.5), and filtering out protons (Section 5.8.6).

### 5.8.1 Local Analysis



**Figure 5.3** Visual representation of the selected cluster in local analysis.

For local analysis, the user will see the attributes of a cluster specified (seen in Figure 5.4) and its visual representation on the pixel screen (seen in Figure 5.3). The pixel screen contains information about the total energy of the clusters (represented by  $E_{\text{layer}}$ ) and the total number of pixels (represented by  $N_{\text{pixels}}$ ). Each pixel is colored according to its energy level (seen on the color bar on the right of the pixel screen). It performs similarly for inspecting coincidence groups.

### 5.8.2 Global Analysis

In a global analysis, the user can browse through clusters interactively in two different ways: by coincidence groups (shown in Figure 5.5) and by a specified



**Figure 5.4** Information regarding the selected cluster for local analysis.

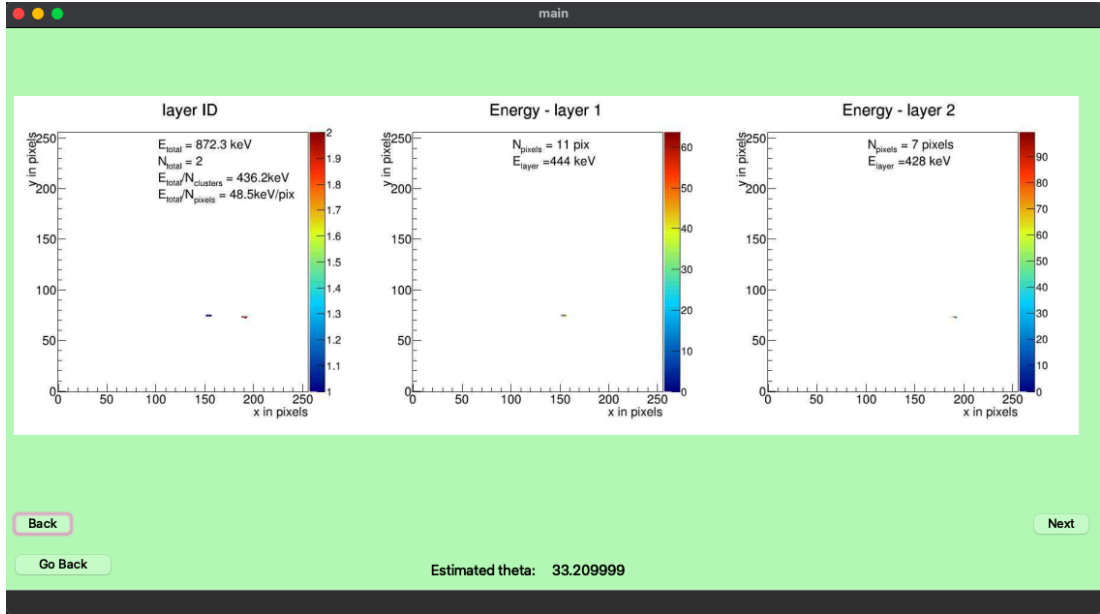
number of clusters per frame. In Figure 5.5, we can see three pixel screens. The first pixel screen contains all the clusters of the coincidence group regardless of which layer they are. The clusters in layer one are shown in blue, and the clusters in layer two are shown in red. The pixel screen also contains information about the total energy of the clusters (represented by  $E_{total}$ ), the number of clusters (represented by  $N_{total}$  or  $N_{clusters}$ ), the average energy per cluster (represented by  $E_{total}/N_{clusters}$ ), and the average energy per pixel (represented by  $E_{total}/N_{pixels}$ , where  $N_{pixels}$  represents the total number of pixels).

The second pixel screen contains information about clusters of the corresponding coincidence group appearing in layer 1. Each pixel is colored according to its energy level (seen on the color bar on the right of the pixel screen). The pixel screen contains information about the total number of pixels (represented by  $N_{pixels}$ ) and the total energy of the clusters in this layer (represented by  $E_{layer}$ ). The same idea applies to the third pixel screen on the right. The only difference is that the third pixel screen contains only clusters appearing in layer 2.

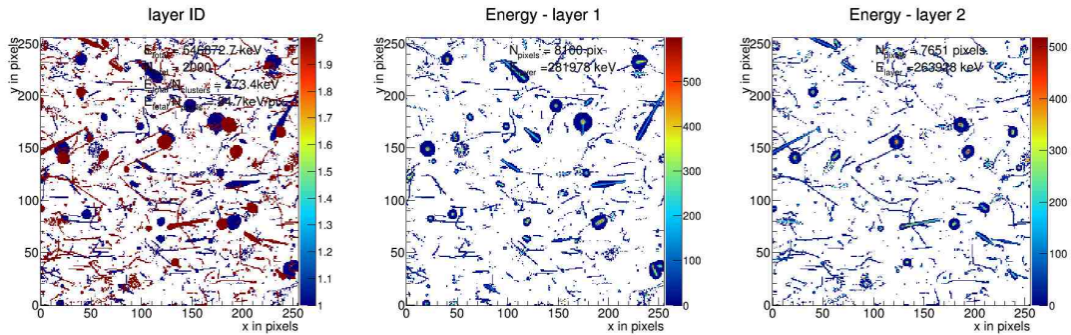
Figure 5.6 shows a visual representation of looking at 2000 clusters per frame. This functionality is particularly useful when the user wants to analyze the density of clusters in some specific region of the pixel screen, which can sometimes tell him/her about the concentration of particles in different regions. This concentration tends to differ more when it comes to analyzing neutrons because of the differences in interactions with the different neutron converters between the detector layers. In interactive viewing for global analysis, for one-to-one matches, the corresponding estimated  $\theta$  is also shown, which is displayed in Figure 5.5.

Additional results of the global analysis are statistics and histograms. Figure 5.7 shows the statistics that will appear after analyzing a dataset of neutrons coming from a fast neutron beam at 120 degrees containing around 80 million clusters. The produced histograms include information about:

- cluster count (Figure 5.8a),
- cluster sizes (Figure 5.8b),



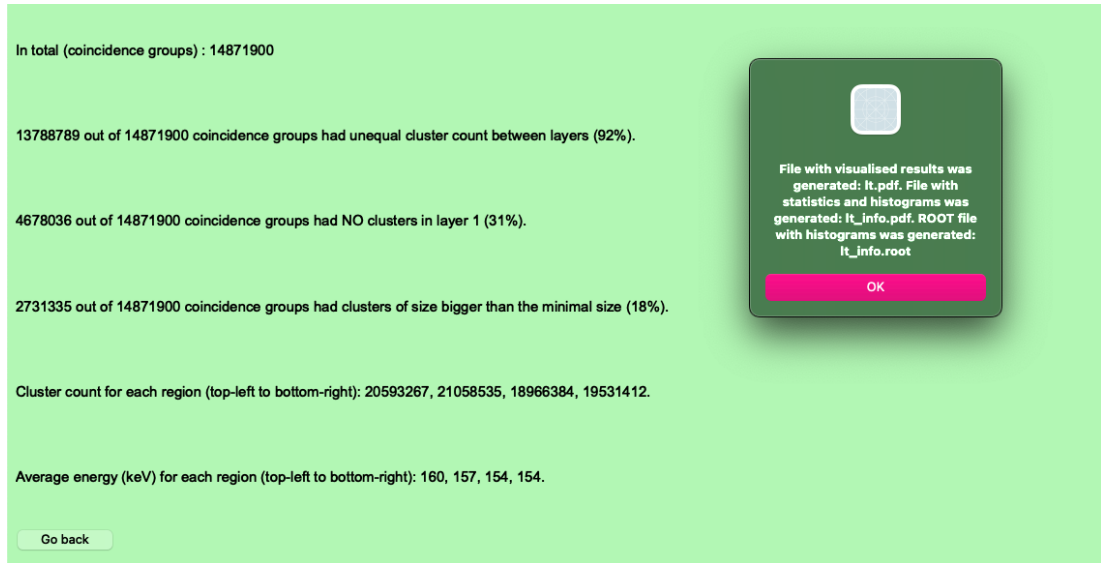
**Figure 5.5** Coincidence group displayed when browsing by coincidence group in global analysis; the estimated  $\theta$  value for this particular pair is also shown.



**Figure 5.6** 2000 clusters displayed when browsing by 2000 clusters per frame (as specified by the user) in global analysis.

- cluster energy per region (Figures 5.9a, 5.9b, 5.9c, and 5.9d),
- original  $\phi$  value (from the ROOT file) and estimated  $\phi$  value using "line of best fit" (Figure 6.3),
- original  $\theta$  value (from the ROOT file) and estimated  $\theta$  value of one-to-one matches in coincidence groups (Figure 5.12).

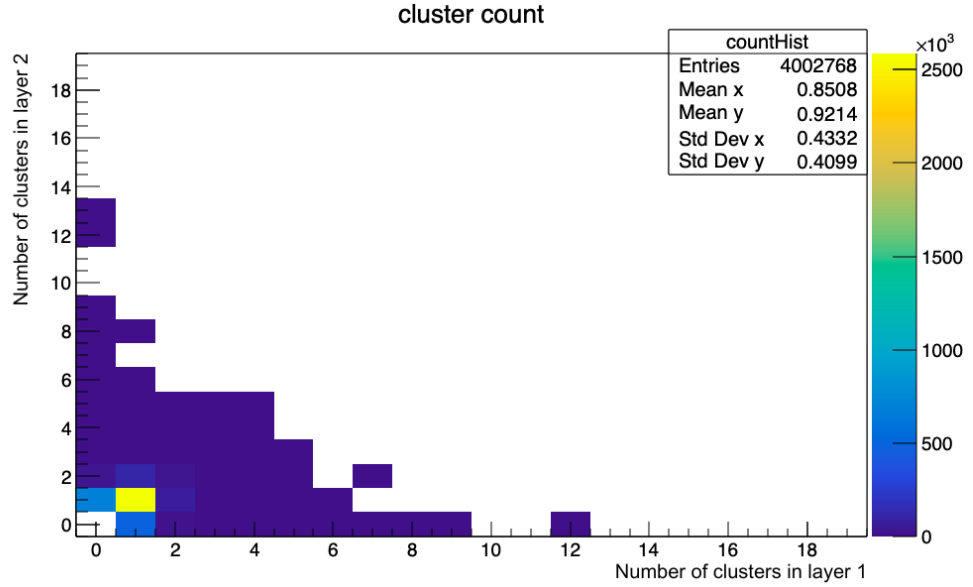
The cluster count and cluster size histograms are two-dimensional histograms, where the axes represent the cluster count or cluster size in layer 1 and layer 2, respectively. In Figure 5.8b, we see that the number of hits is the highest around 10 for each axis, so we can conclude that the majority of clusters in this dataset are of size 10 pixels. We also see some symmetry between the two axes, where the only notable difference is that there are a lot of hits recorded, with the average size being 10 in layer 2 and 0 in layer 1, but not as many the other way around. As we can see, the number of hits is higher along both of the axes as well, and this accounts for coincidence groups where all the clusters were in one layer only.



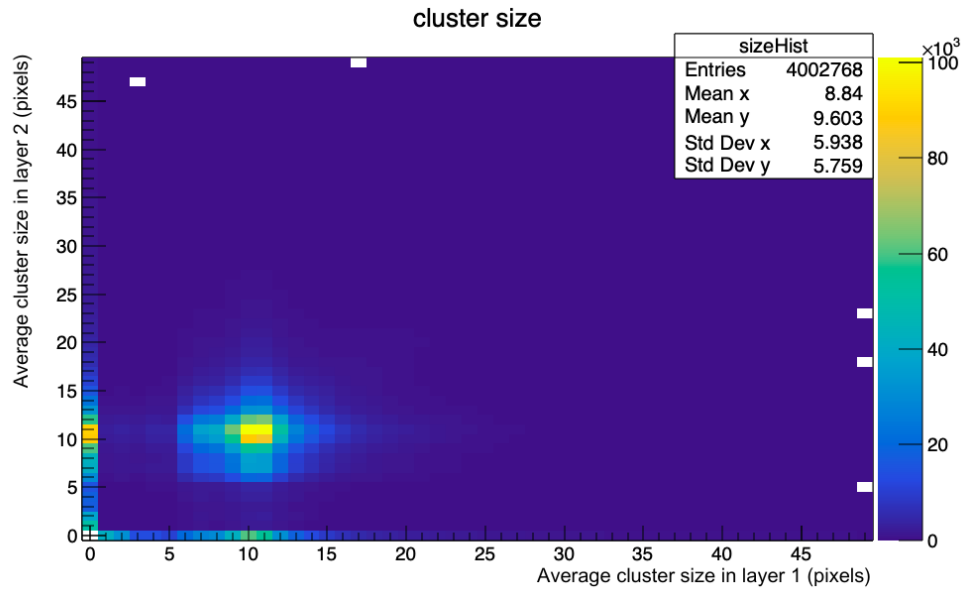
**Figure 5.7** Statistics and general information are shown at the end of analyzing the fast neutron file as a whole in global analysis.

Although the most common value for the average size was around 10 in each layer, it can still get quite big and go up to around 50, although the number of hits will be very low (typically just above 0). That is why we can also see some white cells start appearing in the histogram around the edges, with no hits recorded there at all.

The other histograms are all one-dimensional. In the histogram with cluster energy per region, there are 4 regions defined by the neutron converters, so there are 4 corresponding histograms for each. The  $\phi$  and  $\theta$  histograms are for comparing accuracy between the angle values stored in the dataset and the newly estimated values.

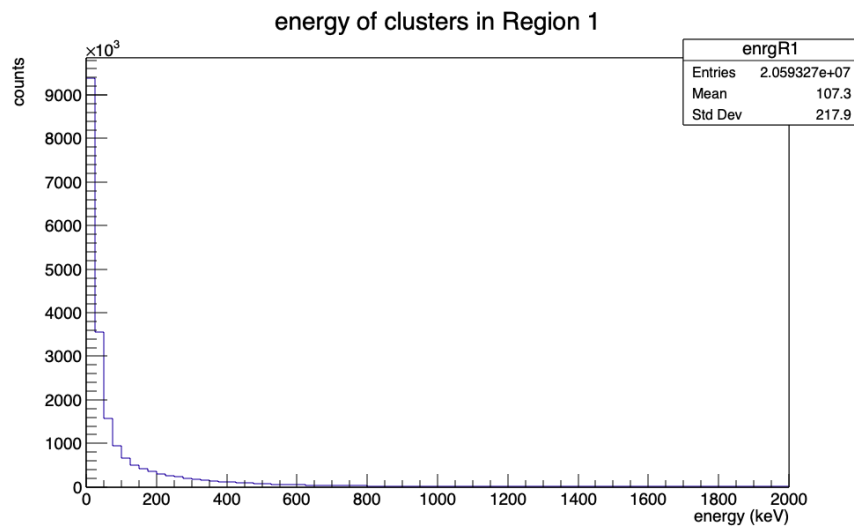


(a) Histogram showing the cluster count within coincidence groups in layers 1 and 2.

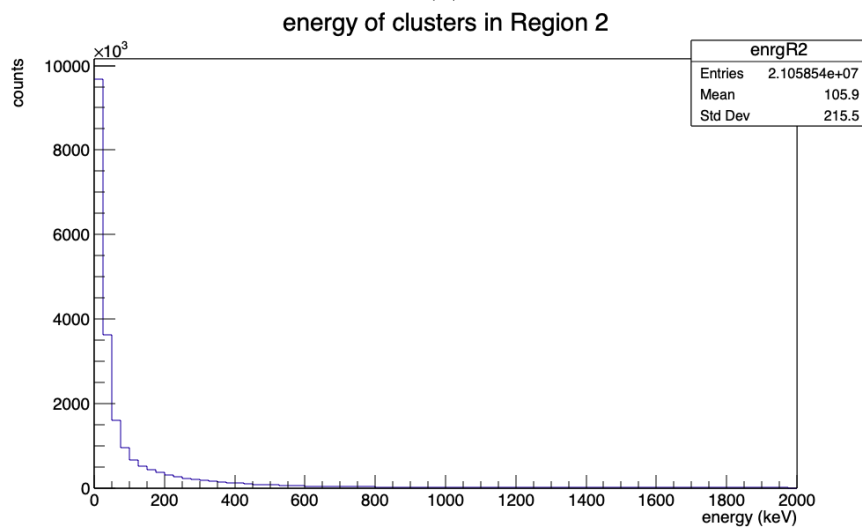


(b) Histogram showing the average cluster sizes within coincidence groups in layers 1 and 2.

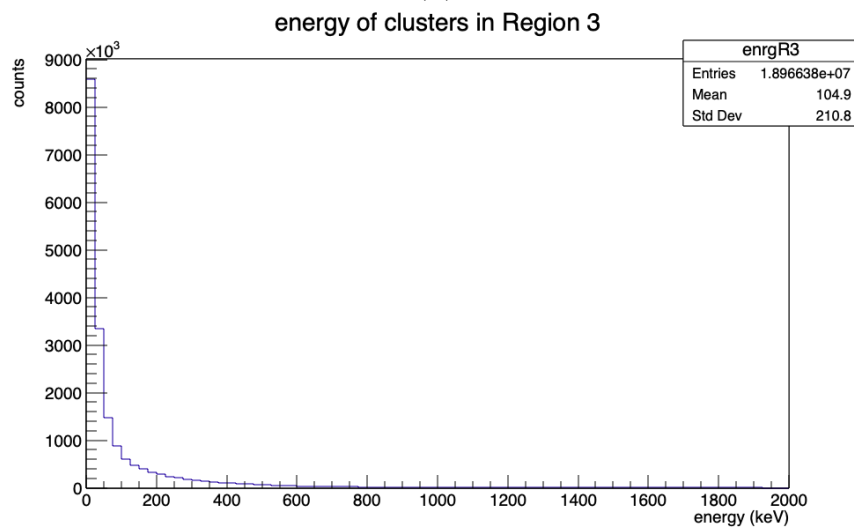
**Figure 5.8** Two-dimensional histograms for cluster count and average cluster size between the layers produced by global analysis of a proton dataset with  $\theta = 34^\circ$ .



(a)

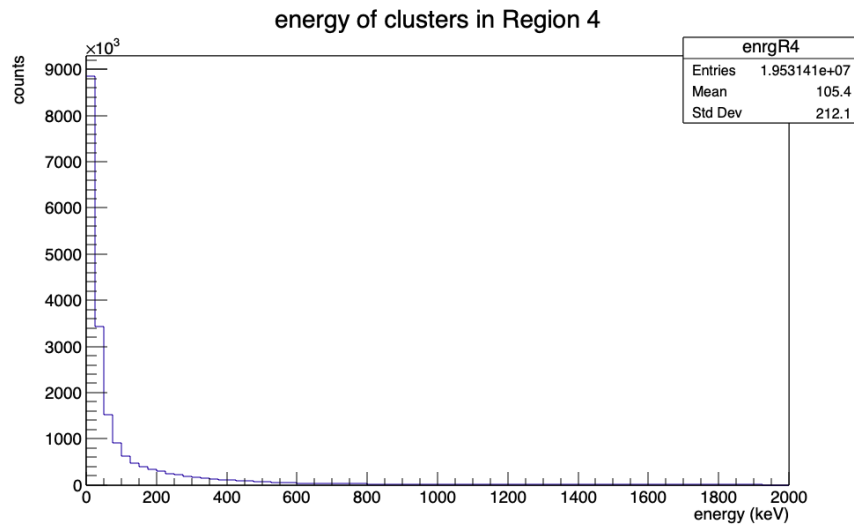


(b)



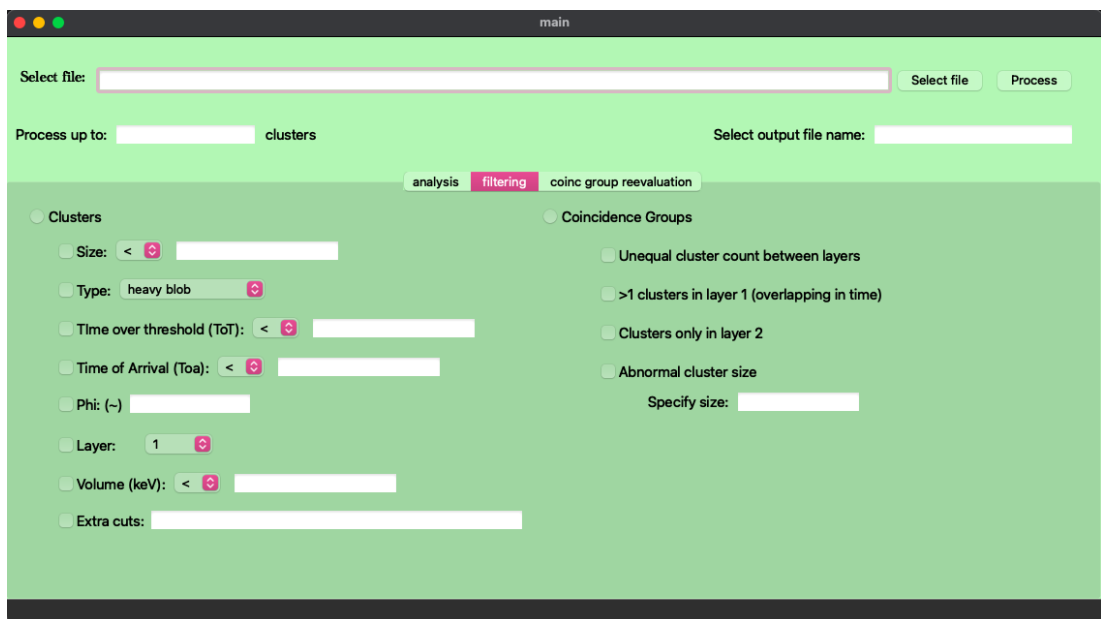
(c)





(d)

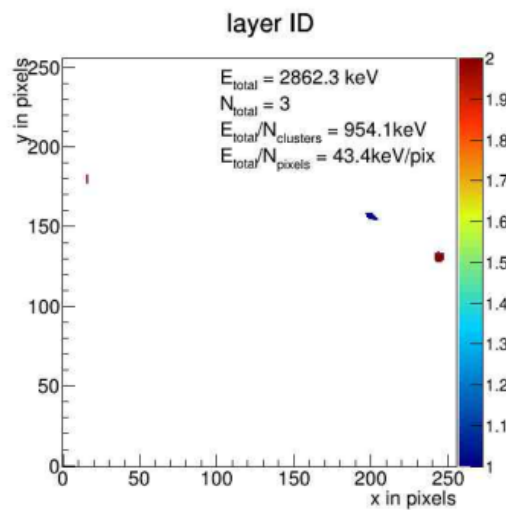
**Figure 5.9** Histograms showing the cluster energy in different detector regions defined by the neutron converters between the layers; this analysis is done for a fast neutron dataset.



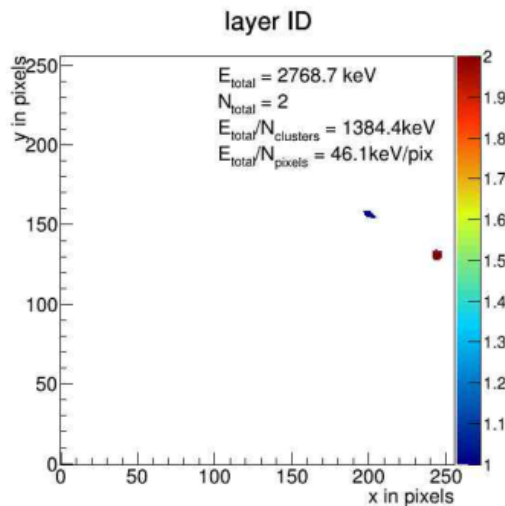
**Figure 5.10** View of the filtering options in the GUI.

### 5.8.3 Filtering

Filtering of clusters or coincidence groups will generate a new ROOT file containing the filtered events, and the user can also view the result interactively. The filtering options are shown in Figure 5.10. For clusters, the filtering is done by applying a cut (a string representing filtering criteria by using logical operators, built-in ROOT functions, and parentheses to create expressions). An example of such a cut can be “`clstrSize > 9`” and in Figure 5.11 we can see a sample coincidence group before and after applying such a cut. The filtering is the same for coincidence groups as well but with different parameters, as described in Section 5.4.



(a) Coincidence group present in the file before applying the cut.



(b) Modified coincidence group after applying the cut where only clusters of size greater than 9 appear.

**Figure 5.11** A sample coincidence group before and after applying the cut “`clstrSize > 9`” during filtering of clusters.

### 5.8.4 Estimating $\theta$

As explained in Section 5.6, we can get more accurate estimations of the angle  $\theta$  for one-to-one matches in proton coincidence groups. When comparing histograms of the original  $\theta$  and the estimated one, the number of entries in the latter is slightly lower than the first one. This is because it only accounts for coincidence groups where  $\theta$  can be estimated (with exactly 1 cluster in each layer). However, this is not a problem as, by doing coincidence group reevaluation on a dataset, we can get even more coincidence groups like that, thus decreasing the difference in the number of entries.

In Figure 5.12, are shown comparisons of histograms of the original  $\theta$  and estimations for proton datasets of  $\theta \in \{0^\circ(+4), 30^\circ(+4), 50^\circ(+4)\}$ . The reason for the (+4) difference is that, in the experiment, the angles were set to a reference position (assumed to be 0 degrees), which, in fact, was already tilted at 4 degrees with respect to the beam. As is seen in the histograms, the mean of the histograms with estimated  $\theta$  values perfectly matches the real  $\theta$  in every example. On the other hand, the histograms with the original  $\theta$  can be close but not as accurate. The biggest difference is for the proton dataset of  $\theta = 4^\circ$  where in the histogram with the original  $\theta$ , the mean is 16.

From these histograms, we can observe much higher accuracy when taking depth into account while computing  $\theta$  for coincidence groups with one-to-one matches in clusters between layers, compared to computing  $\theta$  by only taking into account the attributes available for one single cluster.

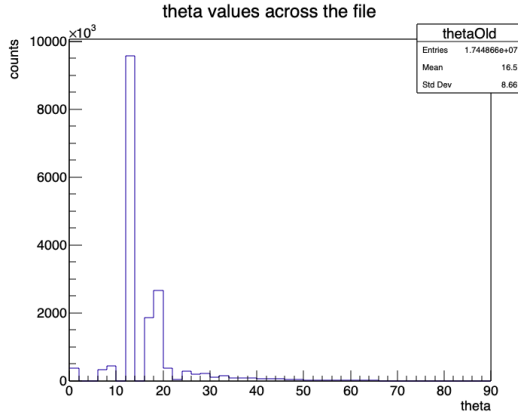
### 5.8.5 Coincidence Group Reevaluation

In coincidence group reevaluation, temporally assigned coincidence groups are reevaluated by additionally considering spatial information. This can be used to distinguish different particles appearing in the same coincidence group. Examples of such usage are in Figures 5.13 and 5.14. On the left of the figures, you can see the coincidence group before it gets reevaluated. In such a state, more than one particles are observed. On the right, it is shown how the reevaluations split the clusters so that they each represent one particle.

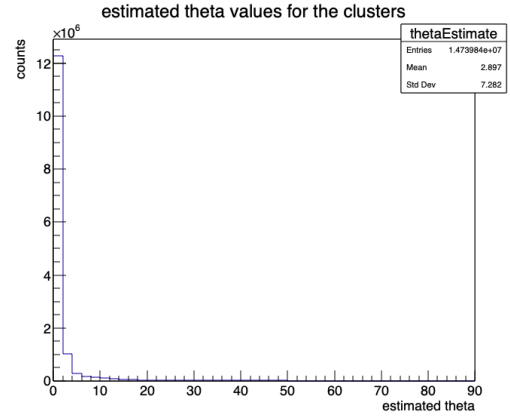
One important thing for reevaluation is adding some constraints on the clusters selected to be inside the reevaluation ellipse. This is because sometimes, based on proximity only, clusters that cannot belong to the same particle because of certain attributes like shape, direction, and so on are still assigned to the same coincidence group. For this reason, there is a constraint check with regard to cluster shape,  $\phi$ , and Euclidean distance based on the original angle  $\theta$ .

The effect of these constraints can be seen in Figure 5.15. In the topmost example 5.15a, the cluster on the left, which was initially (when only using the reevaluation ellipse) considered part of the same coincidence group, is now removed in 5.15b because its distance to the main one in layer 1 differs a lot from the expected distance  $d_{xy}$ . In the middle example 5.15c, the cluster resembling a dot in layer 2 is removed in 5.15d because the cluster length differs too much from the main one in layer 1. In the last example 5.15e at the bottom, the red cluster is first considered to be part of the same coincidence group as the blue one based on proximity, but when we put the constraint on the azimuth (direction)  $\phi$ , it differs a lot from the azimuth of the main cluster so they are not considered to be part

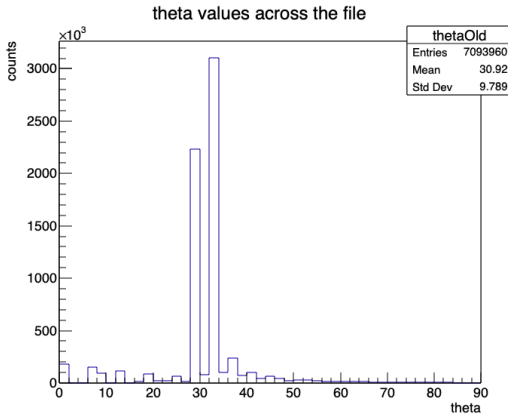
of the same coincidence group anymore in 5.15f.



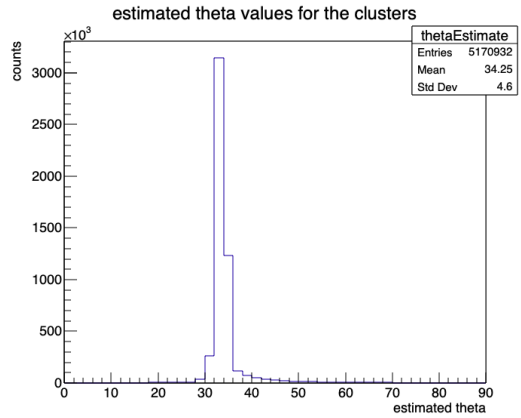
(a) original  $\theta$  saved in the dataset; protons coming from angle  $\theta = 4^\circ$



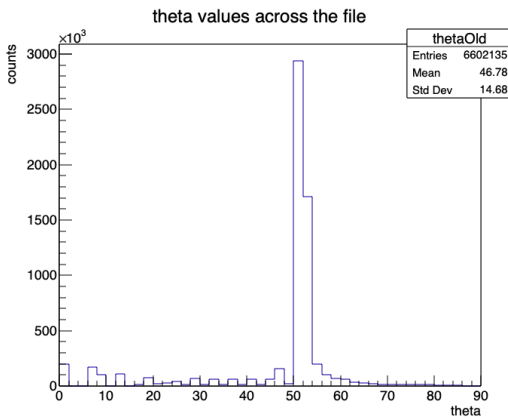
(b) estimated  $\theta$ ; protons coming from angle  $\theta = 4^\circ$



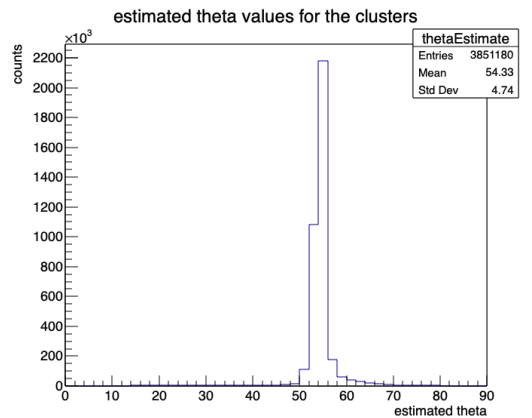
(c) original  $\theta$  saved in the dataset; protons coming from angle  $\theta = 34^\circ$



(d) estimated  $\theta$ ; protons coming from angle  $\theta = 34^\circ$

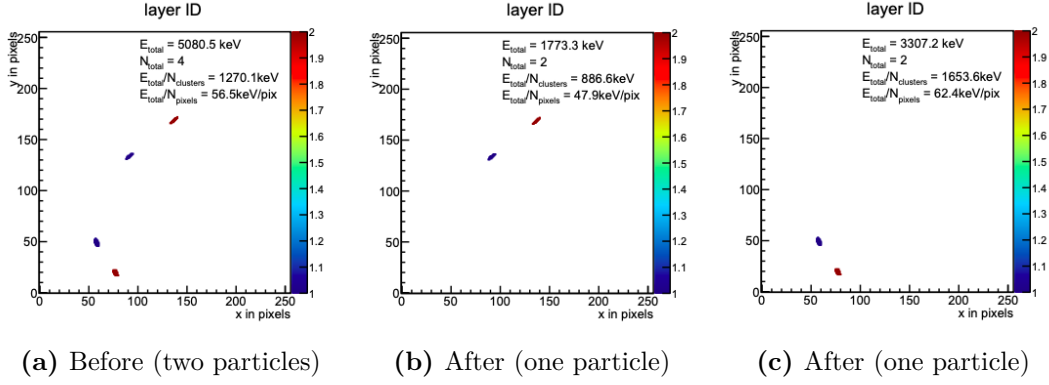


(e) original  $\theta$  saved in the dataset; protons coming from angle  $\theta = 54^\circ$

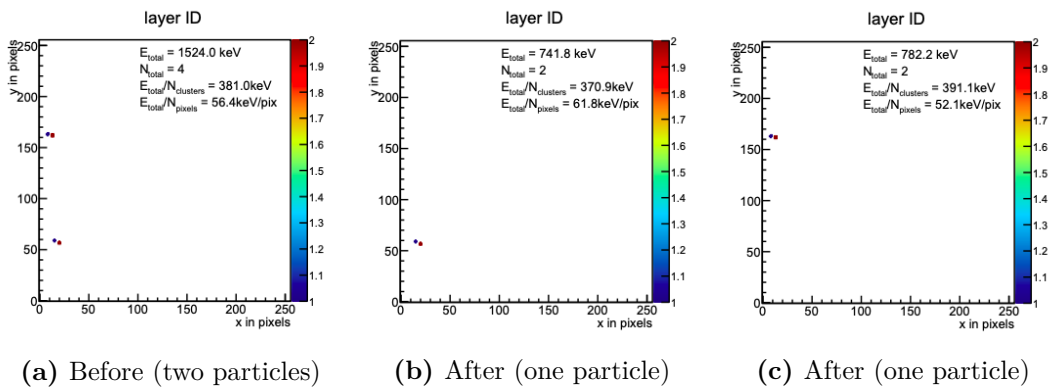


(f) estimated  $\theta$ ; protons coming from angle  $\theta = 54^\circ$

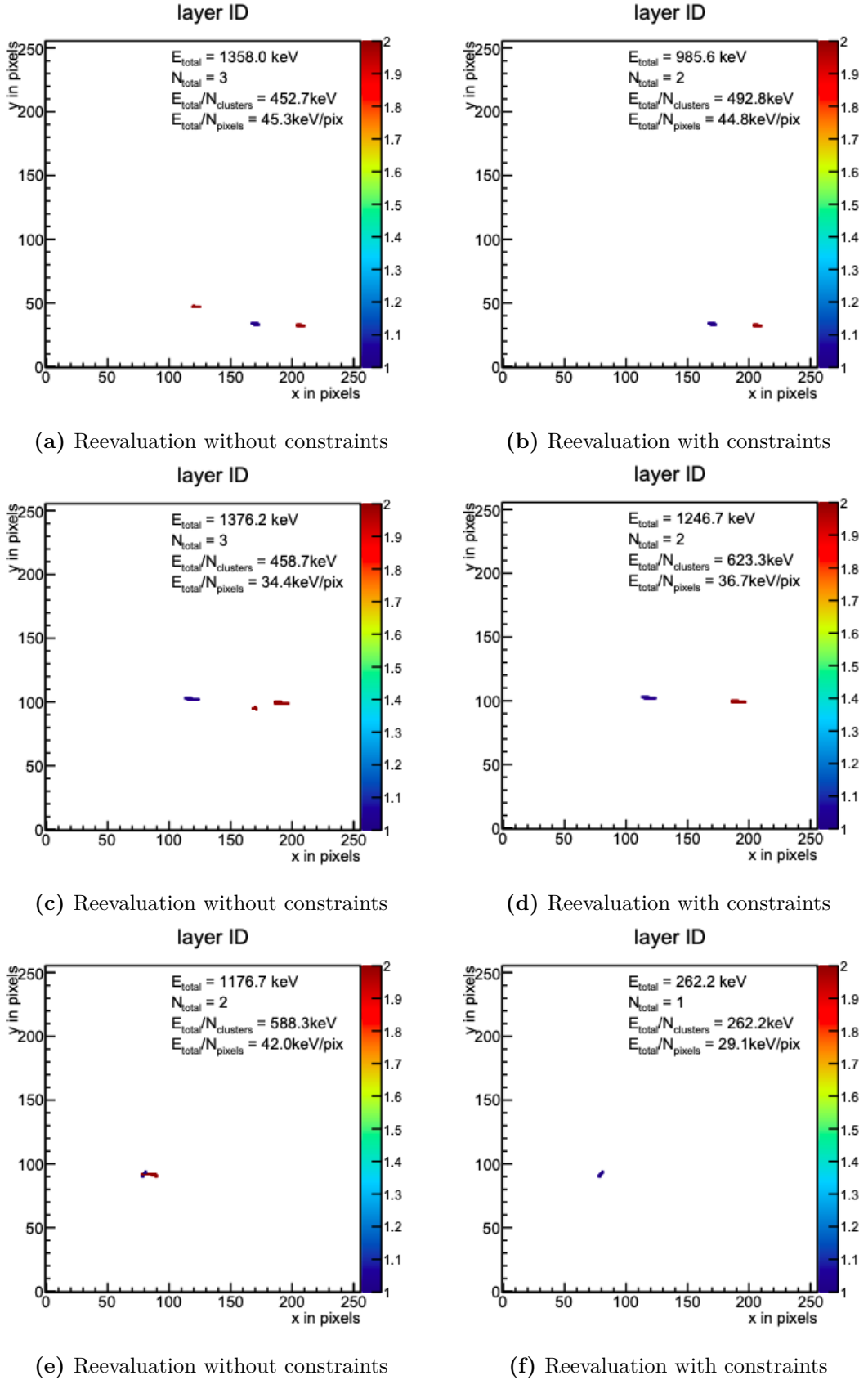
**Figure 5.12** A comparison in accuracy between the original  $\theta$  and the estimated one in various proton files.



**Figure 5.13** A temporally assigned coincidence group being partitioned into 2 new ones using spatial information.



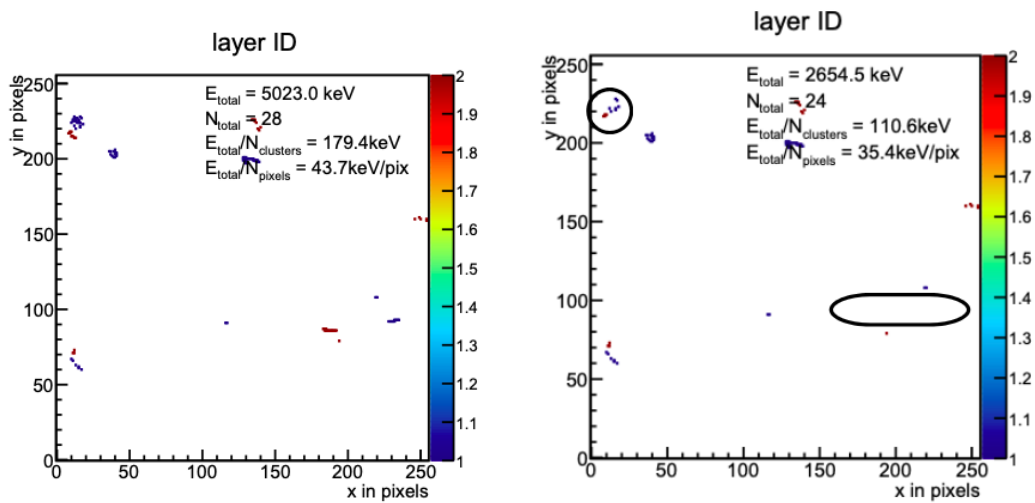
**Figure 5.14** A temporally assigned coincidence group being partitioned into 2 new ones using spatial information.



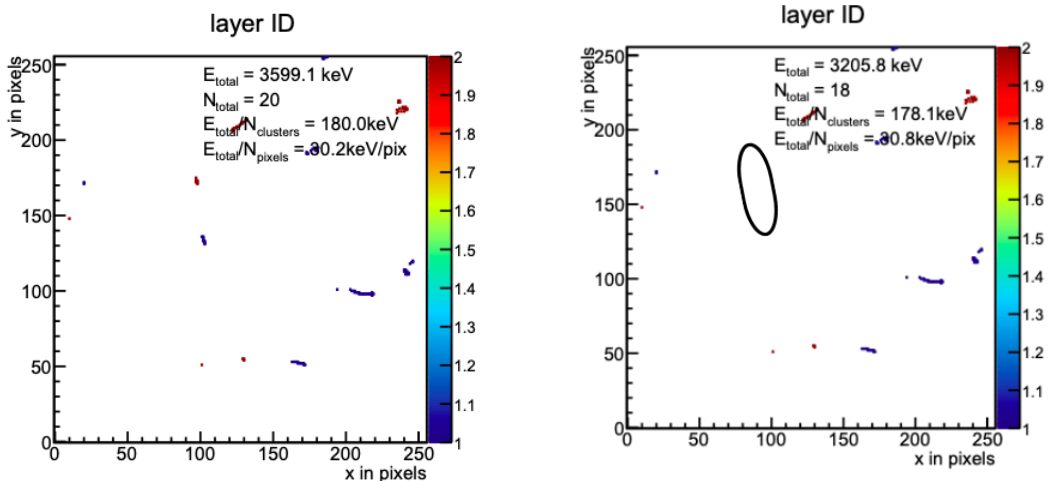
**Figure 5.15** Reevaluation of coincidence groups with and without using constraints on cluster length, azimuth  $\phi$  direction, and Euclidean distance determined by the original angle  $\theta$ .

## 5.8.6 Filtering Out Protons

Filtering out protons is related very closely to coincidence group reevaluation. It is a very beneficial step when it comes to solving a bigger problem – recognizing neutron clusters. In Figure 5.16, it can be seen that from a neutron coincidence group sample, a pair of matching clusters believed to be a proton is removed. In the first example 5.16a, two matching pairs of clusters are removed in 5.16b. One pair is seen in the bottom right region of the screen, and the other one in the upper left corner. In the second example 5.16c, one matching pair of clusters is removed in 5.16d (seen around the center of the screen). A whole new ROOT file of removed proton clusters is generated as a result.



(a) The original neutron coincidence group. (b) After removing two matching pairs of clusters.



(c) The original neutron coincidence group. (d) After removing one matching pair of clusters.

**Figure 5.16** Removing proton particles from two neutron coincidence group samples.



## 5.9 Runtime analysis

We need some test files of various sizes to check the performance of Analyser in different tasks. I will be using five proton files for testing and one neutron file. The proton files are from the ATLAS experiments [27], and each file has a different value of the angle  $\theta$ , which is also determined in the file name. More information about the size of each file and the number of clusters they contain can be found in Table 5.1.

Index	File	File Type	Size	Number of Clusters
1	ATLAS_Unit10_p240MeV_0deg_200V_11cm.root	proton	2.24 GB	17 448 658
2	ATLAS_Unit10_p240MeV_30deg_200V_11cm.root	proton	1.15 GB	7 093 959
3	ATLAS_Unit10_p240MeV_50deg_200V_11cm.root	proton	1.4 GB	6 602 135
4	ATLAS_Unit10_p240MeV_120deg_200V_11cm.root	proton	783.9 MB	3 478 827
5	ATLAS_Unit10_p240MeV_180deg_200V_11cm.root	proton	649.2 MB	5 068 188
6	120_degrees_short.root	neutron	7.81 GB	80 149 597

**Table 5.1** Information about the ROOT files used for testing.

I will define several tasks whose performance I will analyze. I will test the main tasks, such as global analysis, filtering of clusters, filtering of coincidence groups, filtering out protons, coincidence group reevaluation, and training a regression model on a dataset. However, there are also variations of some tasks. For global analysis, I tested it from the GUI and from the command line interface. When testing it from the command line interface, I also tried testing based on whether a visual PDF would be produced in the end to see how that would impact the performance. I also tested the filtering of clusters using various cuts from the GUI and from the command line interface. Apart from the main task, I included a task to just copy the contents of one ROOT file into another, cluster by cluster, to be used as a baseline for comparing performance. A description and an index for each task are found in Table 5.2.

An analysis of the runtime for each task and input file is given in Table 5.3. All of the running times are in seconds. They were measured on a MacBook Air laptop with 8 GB RAM, 1.6 GHz Dual-Core Intel Core i5 processor, running macOS version 12.6.7. Now, we need to describe the results and discuss why some tasks appear to have a much better runtime than others.

Task 1 gives us a baseline for how long a ROOT file would take to be copied into another ROOT file cluster by cluster. The performance aligns closely with

Task Index	Description
1	Copy contents to a new ROOT file
2	Global analysis in the GUI
3	Global analysis in the command line with visual PDF as output
4	Global analysis in the command line with no visual PDF as output
5	Filtering of clusters in the GUI using cut “ $clstrType == 4 \ \&\& \ abs(\phi - 30) < 5 \ \&\& \ layer == 1$ ”
6	Filtering of clusters in the command line interface using cut “ $clusterRoundness > 0.6 \ \&\& \ clstrType > 3$ ”
7	Filtering of coincidence groups to show those that have clusters only in layer two
8	Filtering protons out
9	Coincidence group reevaluation
10	Training a regression model using features <i>clstrRoundness</i> , <i>clstrLength</i> , and <i>clstrType</i>

**Table 5.2** The tasks defined for runtime experiments.

File	Task 1	Task 2	Task 3	Task 4	Task 5	Task 6	Task 7	Task 8	Task 9	Task 10
1	256s	1682s	1960s	117s	70s	2472s	72s	293s	422s	92s
2	130s	760s	1171s	54s	29s	1554s	44s	203s	188s	43s
3	149s	713s	821s	61s	36s	694s	63s	150s	212s	49s
4	85s	440s	385s	31s	18s	404s	33s	123s	119s	24s
5	72s	492s	448s	34s	21s	706s	20s	91s	120s	27s
6	1184s	6128s	2773s	681s	498s	9959s	491s	3926s	None	529s

**Table 5.3** Reported runtime for test files when performing the tasks defined in Table 5.2.

the size of each file. File 6, having the hugest size, takes the longest, and file 5, having the smallest size, takes the shortest.

I should also note the fact that file size does not go hand in hand with the number of clusters. An important factor is the average size of clusters in each file. We can notice this when we compare files 2 and 3, and 4 and 5. File 2 is smaller than file 3, yet it has half a million clusters more. File 5 is also smaller than file 4, yet it has 1.5 million more clusters. This is because the angle  $\theta$  in files 2 and 5 is much closer to the normal to the detector than in files 3 and 4. The angle  $\theta$  influences the shape of proton clusters, where the closer it is to the normal, the smaller the cluster, so it makes sense that the smaller cluster size makes for a

smaller file, even if there are significantly more clusters.

Tasks 2, 3, and 4 all represent global analysis. We can notice at once that the performance of Tasks 2 and 3 is significantly worse than the performance of Task 4. This is mainly because in Tasks 2 and 3, a visual PDF is being produced on the go, and for each cluster, its information must be saved and processed to create the plots used for visual representation. For this reason, Task 4, which only produces output files for statistics and histograms, has a much lower runtime for each input file. Considering this, it is recommended that when the user wants to produce a visual PDF representation, he/she specifies a maximum cluster count for better runtime and also for a PDF whose size is not extreme. For fast processing of a whole file, the user can just perform global analysis via the command line without requesting a visual representation of the whole file. Comparing its performance to the baseline performance of Task 1, it is around twice as good as that of Task 1.

Tasks 5 and 6 represent the filtering of clusters using two different cuts. Task 5 is performed via the GUI, and its performance is very good. Meanwhile, the performance of Task 6, which is performed via the command line, is notably bad. The reason behind this huge difference in performance is the way filtering is done. When using the GUI, the user only filters one file at a time, and for such a procedure, there is a simple built-in method for filtering using ROOT, which takes care of the processing and filtering on its own. When using the command line, however, we must account for more than one input file at once. The only way to continually append to a ROOT file is to process each cluster individually. Checking for each cluster that every filtering condition is satisfied and manually managing the attributes of a ROOT tree when inserting into the new ROOT file creates the performance issue. For this reason, it is recommended that cluster filtering be performed via the GUI unless the user has more than one file to filter at once. The performance of Task 5 is seen to be about 4 times better than that of Task 1.

Task 7 represents the filtering of coincidence groups using the GUI. It performs about 3 times better than Task 1, which we are using as a baseline.

Task 8 represents the filtering of protons out of files. The runtime recorded for each input file is just a bit higher than that of Task 1. The only important input file to consider here is file 6, which represents a neutron file, since we will typically use this functionality on neutron files. The runtime on the neutron file is twice as high as that of Task 1. However, this makes sense, considering that each coincidence group must be looped over and checked twice to know what we need to filter out and what we should preserve.

Task 9 represents the reevaluation of coincidence groups in the GUI. It has not been tested on the neutron file since the reevaluation is intended mainly for protons. The runtime recorded is just above the runtime recorded in Task 1 for a similar reason to the one described in the paragraph above. Hence, this is a reasonably good performance.

Task 10 represents training a regression model on a particular input file. Its performance is reasonably good. The recorded runtime is 2 or 3 times lower than that of Task 1.

## 5.10 Drawbacks

When it comes to coincidence group reevaluation, it will work for all charged particles, although it has been tested mainly on protons. For protons, it will work for all different values of  $\theta$ , except for those where  $\theta$  is close or equal to  $90^\circ$ , since they leave very long tracks parallel to the  $x$ -axis of the detector and have unpredictable behavior. After more careful studying of the patterns of other particles, I am convinced that a similar idea could be implemented in the future as an extension of my implementation to reevaluate other types of particles.

One significant drawback of my implementation is also the fact that the same cluster from layer 2 could visually appear in two different coincidence groups. This happens because of the way I process clusters in a coincidence group. For every cluster in layer 1 that I consider to be a coincidence group of its own, I consider all clusters from layer 2 as possible candidates to be part of the same coincidence group. And if some two clusters in layer 1 are not very close together or have different attributes like length or azimuth  $\phi$  direction, then it will be correctly resolved. However, if they are close, they will have an overlap in possible candidates to be considered part of their coincidence group, so such a cluster might be assigned to more than one proton coincidence group. There is no way to be certain which of those assignments is correct without at least looking at it manually, so the only option for now is to leave it like that.

# 6 Incorporating Machine Learning Techniques

Machine learning techniques can contribute to better data analysis, heuristics, and predictions, offering us deeper insight into the datasets we are working with. I will start this chapter by giving an introduction to machine learning in Section 6.1. In Section 6.2, I will talk about the objectives of this thesis using machine learning. In Section 6.3, I will describe the machine learning techniques I will use, such as linear and quadratic regression. In Section 6.4, I will describe the implementation. Lastly, in Section 6.5, I will conclude by explaining the results of using machine learning methods in my program.

## 6.1 Machine Learning

Generally speaking, machine learning represents a computer program that, given a certain input, can learn from it and make informed decisions or predictions. The input used by the program to learn is called “training data,” and the output (also called a target) that a program produces is generally the knowledge the program gained from learning. If we can also come up with a performance measure, then we can use that to evaluate and train our machine learning model. More precisely, a machine learning model is a model that, given training data, its performance improves the more data you feed it, and we say that such a model learns from the training data [28].

An essential aspect of learning is the ability to predict the outcome on previously unseen input data. This is also known as generalization, and here, we define a generalization error, which tells us how well our model can predict targets of previously unseen data. The original input data (also called features) is often preprocessed to become more useful or fit better into the model. Along with preprocessing, we can choose to do normalization or standardization. Normalization means transforming the data so every feature has normalized values between 0 and 1. Standardization means subtracting the mean and dividing by the standard deviation so the feature values form a distribution similar to that of a standard normal variable.

Based on the training data and what type of output you want to get from it, there are three main types of learning: supervised learning, unsupervised learning, and reinforcement learning. Supervised learning involves learning a function that maps an input to an output based on example input-target pairs, so the training data has the labels or the targets specified. The two main tasks in supervised learning are classification and regression. When we want to assign an input vector to a finite number of discrete categories, then we are dealing with classification. On the other hand, when the output is continuous, the task is called regression.

In unsupervised learning, the training data has no corresponding labels or targets. Common tasks in unsupervised learning are clustering or density estimation. In clustering, you try to categorize similar examples in the data into groups, while in density estimation, you try to determine the distribution of the training data.

Reinforcement learning is about making an agent learn to make informed

decisions that maximize its reward in an environment. The agent typically learns, through trial and error, to come to the optimal solutions.

## 6.2 Program Objectives with Machine Learning

In my application, the goal is to use machine learning for predicting or estimating important properties of the dataset. The datasets we work with are huge, so plenty of learning could be done by exploiting the data we have. Because of this, building a model for a specific task is less beneficial than building a tool that users can use to train their data on whichever features they need and analyze whichever feature they want as a target. Hence, I will propose a multivariate linear and quadratic regressor tool customizable by the user, which I will explain further in Section 6.4.3.

Machine learning (linear or polynomial regression models, in particular) can help us learn important relationships or functions between variables. There are two important tasks in coincidence group reevaluation where I can benefit from such a model: estimating the angle  $\phi$  of a cluster and learning a function between  $\theta$  and the major and minor axes of the reevaluation ellipse. I will explain them in Sections 6.4.1 and 6.4.2.

## 6.3 Techniques to Use and Why

As a general tool for learning relationships between different features of the data, linear and quadratic regression using stochastic gradient descent is a good choice as a start. It is efficient and can be used to get heuristics for certain attributes, e.g., estimating  $\theta$  from cluster length and linearity. Since this model is fairly simple, it will work for now, but more complex models can be implemented for further and more thorough analysis.

For estimating  $\phi$ , we will use the “line of best fit”, which is essentially also just linear regression. Lastly, for setting the parameters of the reevaluation ellipse, using the limited data I collected myself from experimentally playing around with different datasets, I realized the relationship between  $\theta$  and the parameters is not linear so we will use quadratic regression instead, which gives us a perfect fit.

I will describe different methods and how they work in the following sections. In Section 6.3.1, I will introduce linear regression and different implementations of it. I will talk about quadratic regression in Section 6.3.2.

### 6.3.1 Linear Regression

Linear regression models are statistical methods that estimate the relationship between a target variable and one or more input variables by fitting a linear equation to the observed data, where the equation’s coefficients (also called weights) represent the effect of each input variable on the dependent variable (the target) ([28],[29]). Given input example  $\mathbf{x} \in \mathbb{R}^D$ , where  $D \in \mathbb{N}$  represents the dimension of  $\mathbf{x}$ , the goal of regression is to predict the value of one or more continuous targets given the value of a  $D$ -dimensional input vector  $\mathbf{x}$ . A simple

linear regression model computes predictions by estimating the parameters (given below as weights) of a prediction function  $y: \mathbb{R}^D \rightarrow \mathbb{R}$ , where:

$$y(\mathbf{x}; \mathbf{w}, b) = x_1 w_1 + \dots + x_D w_D + b = \sum_{i=1}^D x_i w_i + b = \mathbf{x}^T \mathbf{w} + b, \quad (6.1)$$

where  $\mathbf{w} \in \mathbb{R}^D$  represents the vector of weights for each input feature and  $b \in \mathbb{R}$  represents the bias. Sometimes, instead of dealing with the bias separately, you can extend the input vector by padding a value 1 and encoding the bias as the last weight of  $\mathbf{w}$ . In that case, we have  $\mathbf{w} \in \mathbb{R}^{D+1}$ ,  $\mathbf{x} \in \mathbb{R}^{D+1}$ , and  $y: \mathbb{R}^{D+1} \rightarrow \mathbb{R}$ , defined as:

$$y(\mathbf{x}; \mathbf{w}) = x_1 w_1 + \dots + x_D w_D = \sum_{i=1}^D x_i w_i = \mathbf{x}^T \mathbf{w}. \quad (6.2)$$

Let's define  $\mathbf{X} \in \mathbb{R}^{N \times (D+1)}$  as the training dataset of  $N$  training examples and  $D$  features, padded by a vector  $\mathbf{1} \in \mathbb{R}^N$  to represent the bias, and  $\mathbf{t} \in \mathbb{R}^N$  represents the vector of target values. Finding the weights is done by minimizing some error function that measures how much our model's predictions differ from the real target values. Usually, mean squared error, defined below, is used as such an error function:

$$\text{MSE}(\mathbf{w}) = \frac{1}{N} \sum_{i=1}^N (y(\mathbf{x}_i; \mathbf{w}) - t_i)^2, \quad (6.3)$$

where  $t_i$  represents the real target value of the training example  $\mathbf{x}_i$ . The sum of squares can also be used instead where we take  $\frac{1}{2}$  instead of  $\frac{1}{N}$ :

$$\text{SSE}(\mathbf{w}) = \frac{1}{2} \sum_{i=1}^N (y(\mathbf{x}_i; \mathbf{w}) - t_i)^2, \quad (6.4)$$

Our goal is to minimize this error function, hence getting the weights that give us the most accurate predictions. When minimizing a function, we look for all the points in the function where the partial derivatives (the slopes) with respect to  $\mathbf{w}_j$  are 0. Such points, if they exist, give us local or global extremes of the function. Following the calculations, that leads us to

$$\mathbf{w} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{t}. \quad (6.5)$$

However, this equation only has a solution when  $\mathbf{X}^T \mathbf{X}$  is regular, and we can thus compute its inverse.

There are some techniques that can be used in regression to reduce generalization errors, and these are called regularization. A well-known technique is  $L_2$ -regularization. This technique prefers models with smaller weights and tries to achieve this by giving a penalty to models with larger weights. It uses the following error function to achieve that:

$$\text{SSE}(\mathbf{w}) = \frac{1}{2} \sum_{i=1}^N (y(\mathbf{x}_i; \mathbf{w}) - t_i)^2 + \frac{\lambda}{2} \|\mathbf{w}\|^2, \quad (6.6)$$

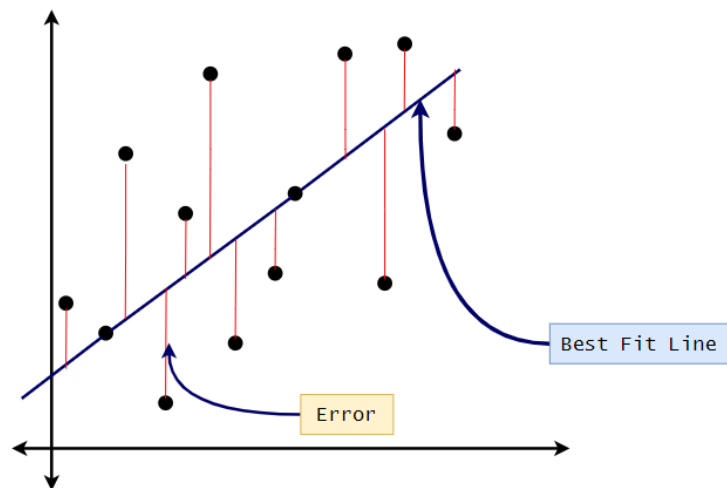
where  $\lambda > 0$  denotes the strength of  $L_2$ -regularization.

Regularization can also help us with the computation of weights. As described earlier, a solution for the weights exists only if  $\mathbf{X}^T\mathbf{X}$  is regular. When  $L_2$ -regularization is included, calculating the partial derivatives of the error function defined in Equation 6.6, gives us the solution:

$$\mathbf{w} = (\mathbf{X}^T\mathbf{X} + \lambda I)^{-1}\mathbf{X}^T\mathbf{t}, \quad (6.7)$$

and for  $\lambda > 0$ ,  $\mathbf{X}^T\mathbf{X} + \lambda I$  is always regular.

### Line of Best Fit



**Figure 6.1** An example of finding a line of best fit to two-dimensional data illustrates the linear relationship between input features and targets, where the dots represent the real targets, the prediction for the same input lies on the constructed line shifted vertically from the real target, and thus the error represents the sum of squares error that will be computed using the difference between the real and predicted targets. [30]

The line of best fit, also known as the regression line, is a straight line that best approximates the relationship between the independent variables (input features) and the dependent variable (target). The line's equation, determined through the linear regression model, minimizes the sum of the squared differences between the observed values and the values predicted by the model. This method involves calculating the slope and intercept of the line in a simple linear regression or the corresponding coefficients (weights) in multiple linear regression. As seen in Figure 6.1, this line may pass through some of the points, none of the points, or all of the points, depending on the data.

### Stochastic Gradient Descent

When dealing with too much data continually being added, sometimes it is better to update the weights incrementally or sequentially. Since the objective was to minimize an error function (let us denote it as  $E(\mathbf{w})$ ), we know that we are looking for  $\underset{\mathbf{w}}{\operatorname{argmin}} E(\mathbf{w})$ . To get to such weight vector  $\mathbf{w}$ , we can update it using gradient descent:

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \nabla_{\mathbf{w}} E(\mathbf{w}), \quad (6.8)$$



where  $\nabla_{\mathbf{w}}E(\mathbf{w})$  is the gradient of the error function at  $\mathbf{w}$  and  $\alpha > 0$  is a learning rate that is the length of a single step we take in every iteration. Stochastic Gradient Descent (SGD) is an iterative algorithm that estimates  $\nabla_{\mathbf{w}}E(\mathbf{w})$  by using a new random example from the training data and updating the weights on each step until convergence.

Following this idea, by using  $L_2$ -regularization and  $SSE(\mathbf{w})$  (defined in equation (6.6)) as the error function, we come to the following algorithm for computing the weights:

---

**Algorithm 1** Linear Regression using SGD

---

**Input** Training data  $\mathbf{X} \in \mathbb{R}^{N \times (D+1)}$ , target vector  $\mathbf{t} \in \mathbb{R}^N$ , learning rate  $\alpha \in \mathbb{R}^+$ ,  $L_2$  strength  $\lambda \in \mathbb{R}^+$

**Output** Weights  $\mathbf{w} \in \mathbb{R}^{D+1}$  that minimize the regularized  $SSE(\mathbf{w})$  error function defined in equation (6.6)

- $\mathbf{w} \leftarrow \mathbf{0}$  or initialize it randomly
  - repeat until convergence (or you decide to stop):
    - sample one example  $\mathbf{x}_i$  from  $\mathbf{X}$  uniformly at random
    - $\mathbf{w} \leftarrow \mathbf{w} - \alpha((\mathbf{x}_i^T \mathbf{w} - t_i)\mathbf{x}_i) - \lambda \mathbf{w}$
- 

### 6.3.2 Quadratic Regression

Polynomial regression is a type of regression where the relationship between the input features  $x$  and the target  $y$  is modeled as an  $n$ -th degree polynomial. In linear regression, when we try to find the expected value of a target  $y$  in terms of  $x$ , typically we use:

$$y = \beta_0 + \beta_1 x + \varepsilon, \quad (6.9)$$

where  $\varepsilon$  represents an unobserved random error with mean zero, and for any increase in  $x$  by some unit,  $y$  changes by  $\beta_1$  units. However, we often cannot express the relationship we are analyzing linearly, and we need to look at polynomials of higher degrees. For  $n$ -th degree polynomial regression, to express the same thing, we would write:

$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \dots + \beta_n x^n + \varepsilon. \quad (6.10)$$

Quadratic regression is a special case of polynomial regression where  $n = 2$ .

### Multivariate Quadratic Regression

When we are dealing with a single input feature, as explained in the above section, we use simple polynomial regression. However, we often have more than one input feature. In this case, we use multivariate polynomial regression [31].

Suppose we have input vector  $\mathbf{x} \in \mathbb{R}^D$ . Now, in order to use the features of  $\mathbf{x}$  for training the model, we must first extend these features to include higher-order terms and additionally include interaction terms between the features. In this case, for a polynomial of degree  $n$ , to train a regression model and make predictions, we need to consider the feature vector

$$\vec{\mathbf{x}} = \left( \underbrace{1}_{\text{degree 0}}, \underbrace{x_1, \dots, x_D}_{\text{degree 1}}, \underbrace{x_1^2, x_1 x_2, \dots, x_1 x_D, x_2^2, \dots, x_D^2}_{\text{degree 2}}, \dots, \underbrace{x_1^n, \dots, x_D^n}_{\text{degree } n} \right)$$

where “degree  $k$ ” denotes the list of all terms of the form  $x_1^{k_1} x_2^{k_2} \cdots x_D^{k_D}$  such that  $\sum_{i=1}^D k_i = n$  for non-negative integers  $k_1, \dots, k_D$ .

In multivariate quadratic regression, you only need to include terms up to degree 2.

## 6.4 Implementation

In the following sections, I will talk about implementation. I will explain how I used the machine learning techniques I explained in the section above to estimate the azimuth angle  $\phi$  in Section 6.4.1 and to compute the parameters of the reevaluation ellipse based on angle  $\theta$  in Section 6.4.2. Lastly, in Section 6.4.3, I will describe the implementation of a customizable multivariate linear or quadratic regressor.

### 6.4.1 Estimating $\phi$

In the datasets I have been working with so far, the azimuth angle  $\phi$  of a cluster is calculated by the following formula:

$$\phi = \left( \arctan \frac{sizeY}{sizeX} \right) \frac{180}{\pi}, \quad (6.11)$$

where  $sizeY$  and  $sizeX$  represent the span of pixels of the cluster in the detector screen in the  $y$ , respectively  $x$ , axis. However, both  $sizeY$  and  $sizeX$  are positive, and this limits the value of  $\phi$  in a range between 0 and 90, which cannot always be true because sometimes we might have clusters that lean towards the bottom, and the direction of such clusters cannot be between 0 and 90. Hence, this is something that needs to be fixed.

What we will use to estimate  $\phi$  is the “line of best fit”. We will use the  $x$ -coordinates of the pixels of a cluster as input and  $y$ -coordinates of the corresponding pixels as the output. Then, we will find a function between them that best describes the cluster, and when we do, the slope of such a function will tell us the value of  $\phi$ , which best fits the function.

The idea, as described in Section 6.3.1, will give us a straight line that best fits the data, so naturally, this estimation will work better for mostly linear clusters.

### 6.4.2 Setting Parameters of the Reevaluation Ellipse Based on $\theta$

Knowing how to set the parameters (specifically the major and the minor of the reevaluation ellipse) is important for getting more accurate results for coincidence group reevaluation. So far, we are only reevaluating protons, and just like we explained in Section 5.5.1, the patterns of protons depend on the angle  $\theta$ . As  $\theta$  grows, the length of a track grows, and so does the expected distance of corresponding clusters between the layers. With a growing distance between layers, we should also increase the major and minor so we make sure the corresponding cluster in layer 2 can be correctly accounted for. Hence, we can naturally assume that the major and the minor are somehow related to the value of  $\theta$ , and we

must find a relationship between them so that we set the parameters correctly no matter what  $\theta$  we are working with.

We were testing the functionality and doing experiments with different values for the major and minor until we got an idea of what to set them to based on  $\theta$ . We figured the following values of major  $a$  and minor  $b$ :

- for  $\theta \in \{0^\circ, 180^\circ\} \implies a = b = 15$ ,
- for  $\theta \in \{30^\circ, 150^\circ\} \implies a = 55, b = 20$ ,
- for  $\theta \in \{50^\circ\} \implies a = 120, b = 25$ .

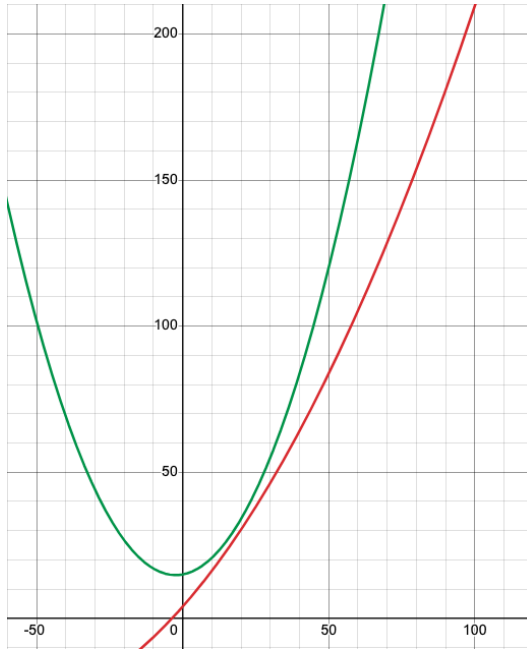
It might seem that the values for the major  $a$  are a bit too high, but that is intentional so that the algorithm is more robust and we do not accidentally leave out clusters that should be considered due to some small change in their patterns.

Now that we had these experimental values that seemed to work for these datasets, we need to figure out a relationship between  $\theta$ , and  $a$  and  $b$ . Looking at the cases above, this relationship does not seem to be linear, so in this case, we go for a quadratic relationship. After using just the few estimations observed above, we get two functions estimating the values for the major and the minor of the reevaluation ellipse. Both these estimating functions are depicted in Figure 6.2 in green. They are called “old” functions there. After further experiments, it turned out that the above estimating functions could sometimes give too high estimations, not enabling the splitting of some of the coincidence groups, where it was possible to select smaller values of the major and the minor manually. In stead of adjusting the estimating functions manually, we decided to use quadratic regression.

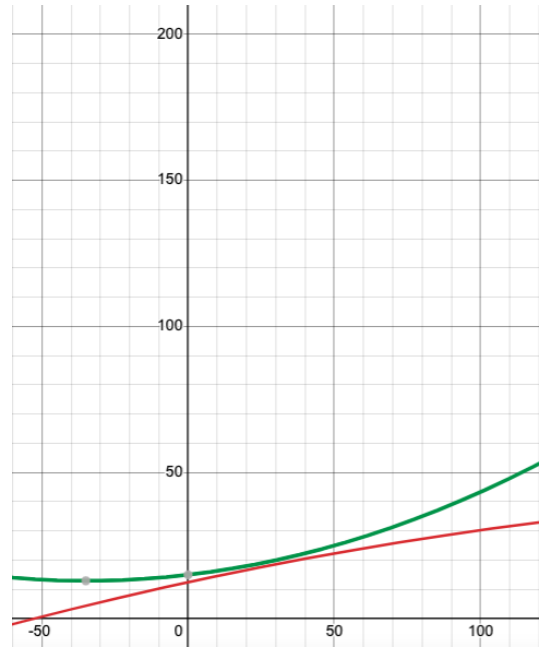
In order to get a good quadratic regression model to approximate the relationships we want, we should have some reliable training data on top of the experiments I mentioned above. Since none of the datasets available are labeled, we have to manually collect and label some training examples that we can use to train our model. This will make for a more solid model with more accurate approximations.

For this reason, I collected information from 135 different samples of coincidence groups that had different values of the angle  $\theta$ . I specifically looked at coincidence groups with one cluster in layer one and one cluster in layer two, which, after manually analyzing them, seemed to be of the same particle. I looked at what was a good value for the major and the minor to make it work, and when recording the information, I added 10 to 15 pixels for a more robust solution.

After collecting this data and re-training the quadratic model, I got functions similar to the previous ones. The differences between the old and new functions that were estimated by the quadratic model are shown in Figure 6.2. In the figure on the left, we can see functions between the angle  $\theta$  and the major of the reevaluation ellipse, where the old one is shown in green while the new one is shown in red. One notable difference is the fact that the new function for the major rises more slowly, giving us lower values for the major than the first one. Similarly, it holds for the function representing the minor in the picture on the right, where the old function is shown in green while the new function is shown in red. This tells us that in the beginning, we were using parameters that were larger than necessary, and we can still perform well by lowering them a little.



(a) Two functions estimated between  $\theta$  and the major of the ellipse. The old one is shown in green; the new one is in red.



(b) Two functions estimated between  $\theta$  and the minor of the ellipse. The old one is shown in green; the new one is in red.

**Figure 6.2** Comparison of functions estimated between the angle  $\theta$  and each parameter of the reevaluation ellipse based on the training data available. The  $x$ -axis represents the value of  $\theta$ , and the  $y$ -axis represents the value of the parameter we are estimating.

### 6.4.3 Customizable Linear and Quadratic regressor

The datasets contain a lot of attributes that give us information about different aspects of a cluster. They are described in detail in Section 2.2. Because there are so many of them, there is a lot of opportunity to learn from this data, as many relationships could be found between them. It is important, that to create the opportunity to learn, we use something simple enough to comply with most of the attributes and efficient enough to give us significant results.

Such a technique that can be used is multivariate linear and quadratic regression using stochastic gradient descent. We have created a tool that uses this technique and is able to describe different properties based on the user's requirements. The user will choose what attributes he/she wants to inspect as independent or input features and what output or dependent feature he/she wants to analyze as a result. The selected features are then normalized to fit in a range between 0 and 1 so that the algorithm is not biased towards certain features with significantly higher values. Then, the data will continually fit into the regression model. Since this model will be available to the user by command line, the user can even use multiple ROOT files as training data. In the end, the user can save the model into a file and use it later to test the model on some test data given in a separate file and analyze its performance on the test set by looking at the RMSE (root mean squared error) it produces. It can also make predictions on given test files and report them to the user in a CSV file.

## 6.5 Results

This section will show results on estimating the azimuth angle  $\phi$  (see Section 6.5.1). In Section 6.5.2, I will further show the accuracy of coincidence group reevaluation before and after implementing machine learning techniques to estimate  $\phi$  and the reevaluation ellipse parameters using  $\theta$  for individual clusters. In Section 6.5.3, I will show examples of using trained regression models.

### 6.5.1 Estimating the Azimuth Angle $\phi$

Estimating the azimuth angle  $\phi$  using the “line of best fit” explained in Section 6.4.1 above is necessary for more accurate coincidence group reevaluations. This will be particularly obvious in the next section. The original values of the azimuth currently recorded in the given datasets are quite limited. They range only between 0 and 90 degrees (an inherent consequence of the pixelation of the detector), thus not representing the full range of direction for a cluster in two-dimensional space. With the new estimations, we get a more precise direction for a cluster.

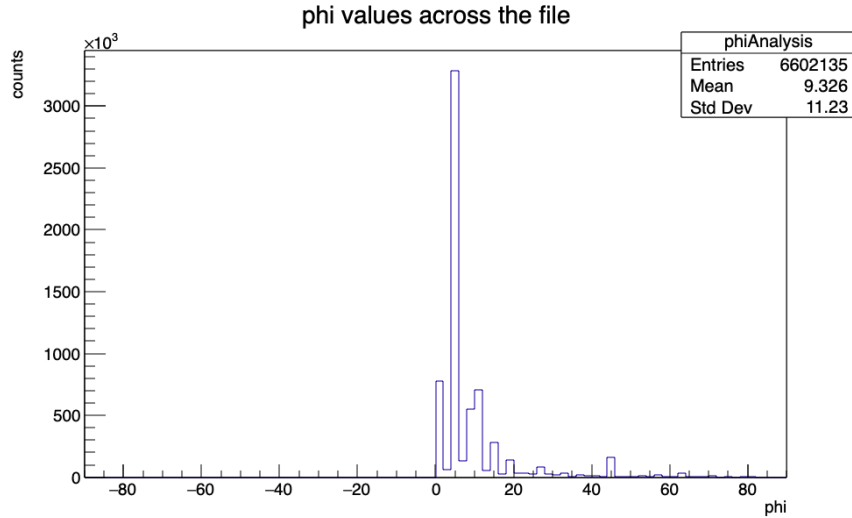
One thing that could be noticed fairly quickly while manually inspecting the proton datasets at my disposal is that most clusters are horizontal but lean slightly to the bottom. However, we do not see such a tendency when looking at histograms of the original azimuth  $\phi$  for a particular proton dataset, which is likely because of the limitation in the range of values. On the other hand, the histogram showing the estimated azimuth angle  $\phi$  represents this tendency very clearly. We can see that the mean degree value of  $\phi$  is also a negative number slightly below 0. This is shown in Figure 6.3.

### 6.5.2 The Effect on Estimations of the Azimuth and Other Parameters on Coincidence Group Reevaluation

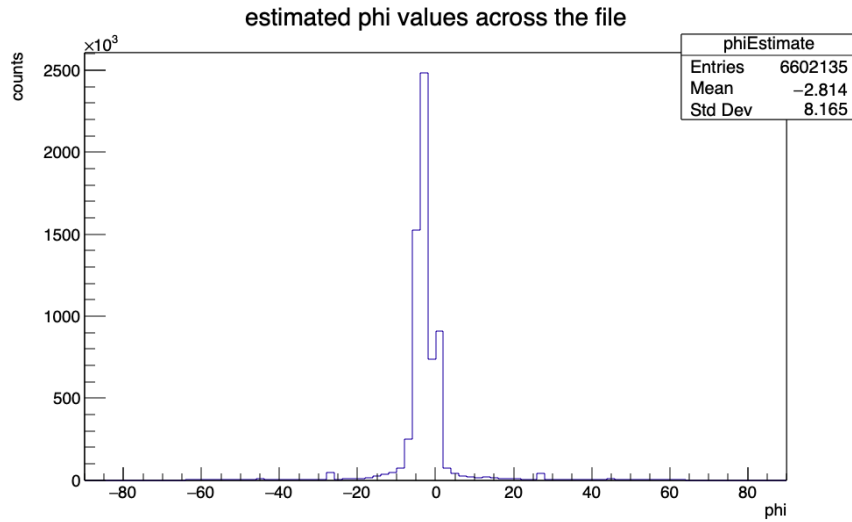
I used proton datasets with  $\theta \in \{0^\circ, 30^\circ, 50^\circ\}$  for testing. The main criteria I used for testing is the number of new one-to-one matches created during reevaluation. The reevaluation improves the accuracy of matching corresponding clusters in both layers. While the coincidence groups in the datasets were considered matching clusters of one particle, the reevaluation splits many of them into independent coincidence groups of matching clusters representing one or more particles. Many of the new coincidence groups created are one-to-one matching pairs, which is the ideal pattern we want to have, considering that a proton left exactly one cluster in each layer it went through. The results can be seen in Figure 6.4.

Experiment 1 shows reevaluation using “line of best fit” to estimate the azimuth  $\phi$  and updating the reevaluation ellipse parameters based on the  $\theta$  angle of each individual cluster. Experiment 2 did the same as Experiment 1 but by using the original  $\phi$  instead of estimating it. Experiment 3 also uses the original  $\phi$  but fixes the parameters of the ellipse in the beginning based on the angle  $\theta$  of the dataset. Experiment 4 estimates the azimuth  $\phi$  using the “line of best fit” but fixes the parameters of the ellipse in the beginning based on the angle  $\theta$ .

From the histograms shown, it can be noticed that Experiments 1 and 4 produce similar values. This is because the only benefit adjusting parameters for



(a) Histogram showing the original value of the azimuth  $\phi$

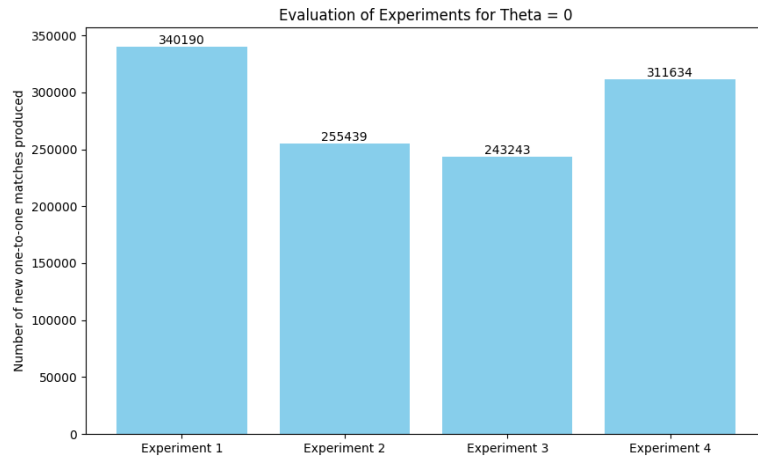


(b) Histogram showing the estimated value of the azimuth  $\phi$

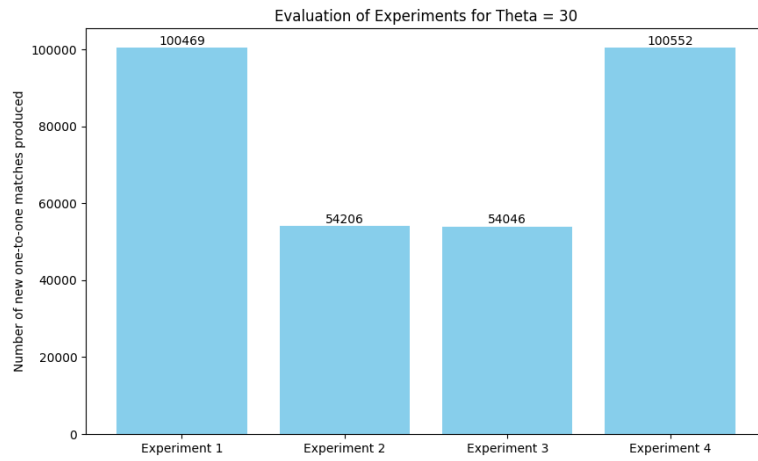
**Figure 6.3** Histograms showing the original and estimated  $\phi$  values in a proton dataset with  $\theta = 50^\circ$ .

each cluster brings is the ability to reevaluate even outliers well (clusters that appear to have a significantly bigger or smaller value of  $\theta$  than the main one). The number of such outliers can be assumed to be significantly small and proportional to the number of clusters in the dataset, hence the similarity in results.

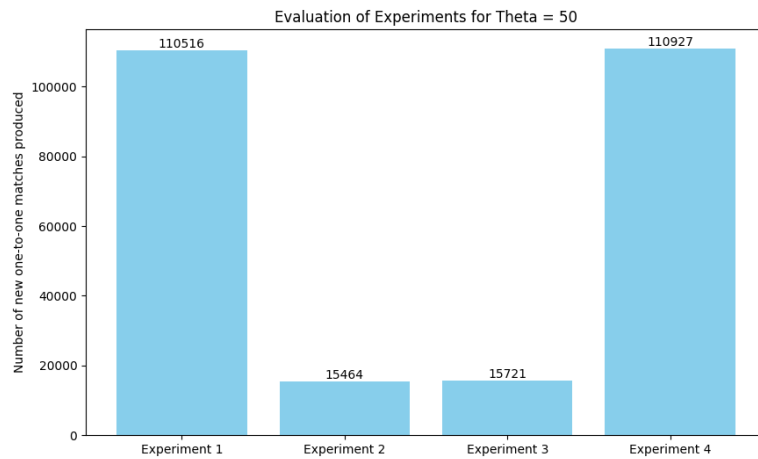
Another thing we can notice from the histograms is that the number of one-to-one matches reduces significantly when we use the original value of the azimuth  $\phi$  instead of estimating it. This indicates that estimating  $\phi$  is helping us go on the right track for more accurate reevaluations.



(a) Proton dataset where main  $\theta = 0^\circ$



(b) Proton dataset where main  $\theta = 30^\circ$

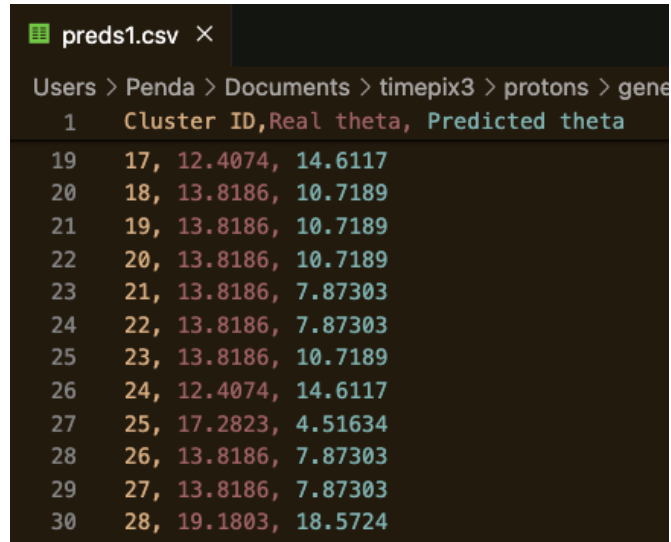


(c) Proton dataset where main  $\theta = 50^\circ$

**Figure 6.4** Histograms showing the number of new one-to-one matches in coincidence groups produced after reevaluation for different proton files.

### 6.5.3 Using Pre-trained Models for Predictions

After training a multivariate linear or quadratic regression model, the user will want to test its performance on different datasets and make predictions. When making predictions, a CSV file with the predicted results will be generated. The CSV file will have three columns: the cluster index, the real output value, and the predicted output value. In Figure 6.5, we can see an example of how the CSV file will look for the first few instances after using a pre-trained multivariate quadratic regression model to predict the value of attribute  $\theta$ .



1	Cluster ID	Real theta	Predicted theta
19	17	12.4074	14.6117
20	18	13.8186	10.7189
21	19	13.8186	10.7189
22	20	13.8186	10.7189
23	21	13.8186	7.87303
24	22	13.8186	7.87303
25	23	13.8186	10.7189
26	24	12.4074	14.6117
27	25	17.2823	4.51634
28	26	13.8186	7.87303
29	27	13.8186	7.87303
30	28	19.1803	18.5724

Figure 6.5 A snippet from a CSV file containing model predictions.



# Conclusion

The goals that were intended to be fulfilled by this application are all completed. In the end, we built a tool for analyzing ROOT datasets for the Timepix3 detector that has multiple functionalities such as local and global analysis, filtering of events, coincidence group reevaluation, filtering out protons, estimation of angles  $\phi$  and  $\theta$ , training regression models, and using the trained models for predictions. Some of the features mentioned were the initial idea since the start of this project, however, after experimenting a lot with the data and finding out about new things we could improve, we ended up with more features than originally thought.

This application can be used by anyone who works with ROOT files (especially those of the Timepix3 detector). It is extensible and can easily be developed further to include additional and more complex functionalities that were not covered in my application. The user can use the tools I have implemented interactively as a GUI application or from the command line to process huge data files, which are very common in high-energy physics. As the application is written in C++, it enables the processing of gigabytes of data in a few minutes on a common laptop computer.

There are some areas that could be worked on further and improved. One such area is the linear or quadratic regression model implemented to find relationships between different attributes of the datasets. There is a quite simple extension that could be made to turn it into a classifier that can be used for detecting events in the ROOT file caused by neutrons.

Another area is coincidence group reevaluation. It currently works mainly for protons because their cluster pattern is more predictable and easy to parameterize. However, I am convinced that after more thorough studying, it can be extended to apply to other types of particles as well.

# Bibliography

1. GRUPEN, Claus; SHWARTZ, Boris. *Particle Detectors, Second Edition* [online]. Cambridge University Press, 2008 [visited on 2024-05-02]. Available from DOI: 10.1017/9781009401531.
2. *Spintharoscope* [online]. [N.d.]. [visited on 2024-05-02]. Available from: <https://orau.org/health-physics-museum/collection/spinthariscopes/index.html>.
3. CHERENKOV, Pavel A. Radiation of particles moving at a velocity exceeding that of light, and some of the possibilities for their use in experimental physics. *Nobel Lectures*. 1958, no. 11.
4. SPIERING, Christian. Cherenkov detectors in astroparticle physics. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* [online]. 2023, vol. 1056, p. 168573 [visited on 2024-05-06]. ISSN 0168-9002. Available from DOI: <https://doi.org/10.1016/j.nima.2023.168573>.
5. HEIJNE, E.H.M.; JARRON, P.; OLSEN, A.; REDAELLI, N. The silicon micropattern detector: A dream? *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* [online]. 1988, vol. 273, no. 2, pp. 615–619 [visited on 2024-05-02]. ISSN 0168-9002. Available from DOI: [https://doi.org/10.1016/0168-9002\(88\)90065-4](https://doi.org/10.1016/0168-9002(88)90065-4).
6. TAMBURRINO, A; CLAPS, G; CONTESSA, GM; PIETROPAOLO, A; CORDELLA, F; DE LEO, V; MONTEREALI, RM; VINCENTI, MA; NIGRO, V; GATTO, R, et al. Thermal neutron detection by means of Timepix3. *The European Physical Journal Plus*. 2023, vol. 138, no. 11, pp. 1–8.
7. DELPIERRE, P. A history of hybrid pixel detectors, from high energy physics to medical imaging. *Journal of Instrumentation* [online]. 2014, vol. 9, no. 05, p. C05059 [visited on 2024-05-02]. Available from DOI: 10.1088/1748-0221/9/05/C05059.
8. SONODA, Minoru; TAKANO, Masao; MIYAHARA, Junji; KATO, Hisatoyo. Computed radiography utilizing scanning laser stimulated luminescence. *Radiology*. 1983, vol. 148, no. 3, pp. 833–838.
9. ROWLANDS, JA. The physics of computed radiography. *Physics in medicine & biology*. 2002, vol. 47, no. 23, R123.
10. PROCZ, S.; AVILA, C.; FEY, J.; ROQUE, G.; SCHUETZ, M.; HAMANN, E. X-ray and gamma imaging with Medipix and Timepix detectors in medical research. *Radiation Measurements* [online]. 2019, vol. 127, p. 106104 [visited on 2024-05-02]. ISSN 1350-4487. Available from DOI: <https://doi.org/10.1016/j.radmeas.2019.04.007>.

11. BALLABRIGA, Rafael; CAMPBELL, Michael; LLOPART, Xavier. Asic developments for radiation imaging applications: The medipix and timepix family. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* [online]. 2018, vol. 878, pp. 10–23 [visited on 2024-05-02]. ISSN 0168-9002. Available from DOI: <https://doi.org/10.1016/j.nima.2017.07.029>. Radiation Imaging Techniques and Applications.
12. LLOPART, X.; ALOZY, J.; BALLABRIGA, R.; CAMPBELL, M.; CASANOVA, R.; GROMOV, V.; HEIJNE, E.H.M.; POIKELA, T.; SANTIN, E.; SRISKARAN, V.; TLUSTOS, L.; VITKOVSKIY, A. Timepix4, a large area pixel detector readout chip which can be tiled on 4 sides providing sub-200 ps timestamp binning. *Journal of Instrumentation* [online]. 2022, vol. 17, no. 01, p. C01044 [visited on 2024-05-02]. Available from DOI: 10.1088/1748-0221/17/01/C01044.
13. SMOLYANSKIY, P.; BACAK, M.; BERGMANN, B.; BROULÍM, P.; BURIAN, P.; ČELKO, T.; GARVEY, D.; GUNTHOTI, K.; INFANTES, F.G.; MÁNEK, P.; MANNA, A.; MRÁZ, F.; MUCCIOLA, R.; POSPÍŠIL, S.; SITARZ, M.; URBAN, O.; VYKYDAL, Z.; WENDER, S.A. A two-layer Timepix3 stack for improved charged particle tracking and radiation field decomposition. *Journal of Instrumentation* [online]. 2024, vol. 19, no. 02, p. C02016 [visited on 2024-05-02]. Available from DOI: 10.1088/1748-0221/19/02/C02016.
14. BERGMANN, B.; CAICEDO, I.; LEROY, C.; POSPISIL, S.; VYKYDAL, Z. ATLAS-TPX: a two-layer pixel detector setup for neutron detection and radiation field characterization. *Journal of Instrumentation* [online]. 2016, vol. 11, no. 10, P10002 [visited on 2024-05-02]. Available from DOI: 10.1088/1748-0221/11/10/P10002.
15. BERGMANN, Benedikt; GOHL, Stefan; GARVEY, Declan; JELÍNEK, Jindřich; SMOLYANSKIY, Petr. Results and Perspectives of Timepix Detectors in Space—From Radiation Monitoring in Low Earth Orbit to Astroparticle Physics. *Instruments* [online]. 2024, vol. 8, no. 1 [visited on 2024-05-06]. ISSN 2410-390X. Available from DOI: <https://doi.org/10.3390/instruments8010017>.
16. *ROOT - Data Analysis Framework* [online]. [N.d.]. [visited on 2024-05-02]. Available from: <https://root.cern/>.
17. *Danish Center for Particle Therapy in Aarhus* [online]. [N.d.]. [visited on 2024-05-06]. Available from: <https://www.en.auh.dk/departments/the-danish-centre-for-particle-therapy/>.
18. *Los Alamos Neutron Science Center* [online]. [N.d.]. [visited on 2024-05-06]. Available from: <https://lansce.lanl.gov/>.
19. MEDUNA, Lukáš; BERGMANN, Benedikt; BURIAN, Petr; MÁNEK, Petr; POSPÍŠIL, Stanislav; SUK, Michal. *Real-time Timepix3 data clustering, visualization and classification with a new Clusterer framework* [online]. 2019. [visited on 2024-04-21]. Available from arXiv: 1910.13356 [physics.ins-det].

20. ČELKO, Tomáš. *Clustering hits and predictions in data from TimePix3 detectors*. [online]. 2023. [visited on 2024-04-25]. Available from: <https://dspace.cuni.cz/handle/20.500.11956/184038>. MA thesis. Charles University, Faculty of Mathematics and Physics, Prague.
21. MEDUNA, Lukáš. *Detecting elementary particles with Timepix3 detector*. [online]. 2019. [visited on 2024-04-25]. MA thesis. Charles University, Faculty of Mathematics and Physics, Prague.
22. ČELKO, Tomáš. *Support for annotating and classifying particles detected by Timepix3*. [online]. 2021. [visited on 2024-04-25]. Available from: <https://dspace.cuni.cz/handle/20.500.11956/148280>. Bachelor's thesis. Charles University, Faculty of Mathematics and Physics, Prague.
23. MÁNEK, P.; BURIAN, P.; DAVID-BOSNE, E.; SMOLYANSKIY, P.; BERGMANN, B. Track Lab: extensible data acquisition software for fast pixel detectors, online analysis and automation. *Journal of Instrumentation* [online]. 2024, vol. 19, no. 01, p. C01008 [visited on 2024-05-02]. Available from DOI: 10.1088/1748-0221/19/01/C01008.
24. MÁNEK, Petr; BEGERA, Jakub; BERGMANN, Benedikt; BURIAN, Petr; JANEČEK, Josef; POLANSKY, Stepan; POSPÍŠIL, Stanislav; SUK, Michal. Software system for data acquisition and analysis operating the ATLAS-TPX network. In: *2017 International Conference on Applied Electronics (AE)* [online]. 2017, pp. 1–4 [visited on 2024-04-21]. Available from DOI: 10.23919/AE.2017.8053593.
25. *QT* [online]. [N.d.]. [visited on 2024-05-02]. Available from: <https://www.qt.io/download-open-source>.
26. *Ellipse parameters* [online]. [N.d.]. [visited on 2024-05-02]. Available from: <https://desktop.arcgis.com/en/arcmap/latest/tools/data-management-toolbox/table-to-ellipse.htm>.
27. *ATLAS experiments* [online]. [N.d.]. [visited on 2024-05-02]. Available from: <https://home.cern/science/experiments/atlas>.
28. BISHOP, Christopher. *Pattern Recognition and Machine Learning*. Springer, 2006. Available also from: <https://www.microsoft.com/en-us/research/publication/pattern-recognition-machine-learning/>.
29. STRAKA, Milan [online]. [N.d.]. [visited on 2024-04-21]. Available from: <https://ufal.mff.cuni.cz/courses/npfl129/2324-winter#lectures>.
30. *Line of best fit* [online]. [N.d.]. [visited on 2024-05-02]. Available from: <https://www.programsbuzz.com/article/best-fit-line>.
31. CASTRO, Yohann De; GAMBOA, Fabrice; HENRION, Didier; HESS, Roxana; LASSERRE, Jean-Bernard. APPROXIMATE OPTIMAL DESIGNS FOR MULTIVARIATE POLYNOMIAL REGRESSION. *The Annals of Statistics* [online]. 2019, vol. 47, no. 1, pp. 127–155 [visited on 2024-05-08]. ISSN 00905364, ISSN 21688966. Available from: <https://www.jstor.org/stable/26581842>.

# List of Figures

1.1	The active layer of a Timepix3 detector bump-bonded to the chip of the Timepix3 detector where the signal is processed (source [6])	10
1.2	Composition of the two-layer detector (Reproduced from [14] under CC 4.0.). (a) Charged particle tracking and neutron detection (b) Layout of the converters . . . . .	13
1.3	Design of the two-layer detector (Reproduced from [13] under CC 4.0.). (a) Side view of the fully assembled device; (b) Front view of the fully assembled design . . . . .	13
1.4	Cluster types classification (Reproduced from [15] under CC 4.0.)	14
2.1	Experimental setup for the proton and fast neutron measurements.	18
2.2	(a) and (b) represent two different coincidence groups for neutrons detected in a fast neutron beam where the long tracks observed are likely electrons or photons; (c) coincidence group for a proton approaching the detector at a perpendicular angle; (d) coincidence group for a proton approaching the detector at an angle of $50^\circ$ away from the normal, where we can observe that two different particles appear as one coincidence group. . . . .	19
5.1	Visual representation of a reevaluation ellipse where we see how the azimuth ( $\phi$ ) is defined, along with the major and the minor axes (image source [26]) . . . . .	29
5.2	How sample reevaluation ellipses look like for different types of protons. . . . .	30
5.3	Visual representation of the selected cluster in local analysis. . . . .	35
5.4	Information regarding the selected cluster for local analysis. . . . .	36
5.5	Coincidence group displayed when browsing by coincidence group in global analysis; the estimated $\theta$ value for this particular pair is also shown. . . . .	37
5.6	2000 clusters displayed when browsing by 2000 clusters per frame (as specified by the user) in global analysis. . . . .	37
5.7	Statistics and general information are shown at the end of analyzing the fast neutron file as a whole in global analysis. . . . .	38
5.8	Two-dimensional histograms for cluster count and average cluster size between the layers produced by global analysis of a proton dataset with $\theta = 34^\circ$ . . . . .	39
5.9	Histograms showing the cluster energy in different detector regions defined by the neutron converters between the layers; this analysis is done for a fast neutron dataset. . . . .	41
5.10	View of the filtering options in the GUI. . . . .	41
5.11	A sample coincidence group before and after applying the cut “clstrSize > 9” during filtering of clusters. . . . .	42
5.12	A comparison in accuracy between the original $\theta$ and the estimated one in various proton files. . . . .	45

5.13	A temporally assigned coincidence group being partitioned into 2 new ones using spatial information. . . . .	46
5.14	A temporally assigned coincidence group being partitioned into 2 new ones using spatial information. . . . .	46
5.15	Reevaluation of coincidence groups with and without using constraints on cluster length, azimuth $\phi$ direction, and Euclidean distance determined by the original angle $\theta$ . . . . .	47
5.16	Removing proton particles from two neutron coincidence group samples. . . . .	48
6.1	An example of finding a line of best fit to two-dimensional data illustrates the linear relationship between input features and targets, where the dots represent the real targets, the prediction for the same input lies on the constructed line shifted vertically from the real target, and thus the error represents the sum of squares error that will be computed using the difference between the real and predicted targets. [30] . . . . .	56
6.2	Comparison of functions estimated between the angle $\theta$ and each parameter of the reevaluation ellipse based on the training data available. The $x$ -axis represents the value of $\theta$ , and the $y$ -axis represents the value of the parameter we are estimating. . . . .	60
6.3	Histograms showing the original and estimated $\phi$ values in a proton dataset with $\theta = 50^\circ$ . . . . .	62
6.4	Histograms showing the number of new one-to-one matches in coincidence groups produced after reevaluation for different proton files. . . . .	63
6.5	A snippet from a CSV file containing model predictions. . . . .	64
A.1	The analysis options of the application. . . . .	75
A.2	Interactive view of a coincidence group. . . . .	76
A.3	Options to choose during the global analysis of a file. . . . .	76
A.4	Message informing you that the global analysis is finished and the output files are generated. . . . .	76
A.5	An example of a filtered coincidence group containing a cluster of size bigger than the minimal size specified (25 pixels). . . . .	77
A.6	The filtering options available. . . . .	79
A.7	The cluster types available. . . . .	80
A.8	An example of a coincidence group with clusters satisfying the cut “ <code>clstrRoundness&gt;0.6 &amp;&amp; clstrType&gt;3</code> ” during filtering of clusters. . . . .	81
A.9	An example of a filtered coincidence group with clusters only in layer 2. . . . .	81
A.10	Possibilities for obtaining filtering clusters or coincidence groups. . . . .	82
A.11	Messages informing you that the filtered ROOT and PDF files have been generated. . . . .	82
A.12	The options for coincidence group reevaluation. . . . .	83
A.13	Output possibilities for reevaluating coincidence groups. . . . .	84
A.14	Messages informing you that the reevaluated ROOT and PDF files have been generated. . . . .	84
A.15	A snippet from a CSV file containing model predictions. . . . .	89

# List of Tables

5.1	Information about the ROOT files used for testing. . . . .	49
5.2	The tasks defined for runtime experiments. . . . .	50
5.3	Reported runtime for test files when performing the tasks defined in Table 5.2. . . . .	50

# A User Documentation

This section explains how to run and use the application developed as part of this thesis. The application is called Analyser, and it is built to offer the processing of ROOT files for various purposes, such as analysis, filtering, or coincidence group reevaluation. It offers a graphical user interface and command line interface for batch file processing. It is supported on MacOS and Linux operating systems. In Section A.1, I will list the dependencies this program relies on. I will describe the installation steps in Section A.2. In Section A.3, I will explain how to run the application and the functionalities offered in the GUI and command line interface.

## A.1 Dependencies

The dependencies required for this application are:

- Qt Framework (either Qt5 or Qt6), with specific components required: Widgets, Pdf,
- ROOT (version 6.30.02),
- Eigen (version 3.4.0).

I should note that some of the dependencies listed above will take a long time to install. It is, in particular, the case for Qt. When I tested and tried installing it in Linux, the installation took almost an hour.

## A.2 Installation

To install Analyser, you must first have all the required dependencies installed. Then, if you are running it on MacOS or Linux, you should follow these steps:

1. Clone the git repository of the project:

```
git clone https://gitlab.mff.cuni.cz/smajljap/timepix3gui.git
```

2. Go to the cloned directory where the source files are.
3. While inside the directory with the source files, create a new directory for building the application and go inside the new directory:

```
mkdir build && cd build
```

4. Once inside the *build* directory, set an environment variable called `EIGEN3_INCLUDE_DIR` to the path of the Eigen dependency on your computer (cmake checks for this environment variable to find the Eigen dependency):

```
echo 'export EIGEN3_INCLUDE_DIR=/path/to/eigen' >> ~/.bashrc
```

5. Update the current terminal window to include the changes just made to the `~/.bashrc` file:



```
source ~/.bashrc
```

6. Run *cmake* and specify the paths to the dependencies so it can find them:

```
cmake .. -DCMAKE_BUILD_TYPE=Release  
-DCMAKE_PREFIX_PATH=/path/to/qt6
```

7. Note: When specifying the path to your Qt installation:

- (a) Find the Qt installation directory.
- (b) Inside, select the directory named after the version of Qt you installed, e.g., 6.5.3.
- (c) Inside the version directory, you have multiple directories representing different architectures.
- (d) Select the one compatible with your system, e.g. “gcc\_64” for Linux or “macos” for MacOS, and take the path to that directory as the path to Qt.

8. Run *make* to build the executable:

```
make
```

9. MacOS users, see step 10. Linux users see step 11.

10. In the *build* directory, a folder called *Analyser* is created. Now, there are two options:

- (a) If you want to use the GUI, you can do so by clicking on it or running:

```
open Analyser.app
```

- (b) If you want to get to the executable directly, you can redirect to:

```
cd Analyser.app/Contents/MacOS
```

and here you can run the executable as:

```
./Analyser
```

If you do not provide any command line arguments, the GUI will open, and if you do, then you will be running the application via the command line for batch processing.

11. In the *build* directory, you will now see the executable *Analyser*. You can run it as:

```
./Analyser
```

If you do not provide any command line arguments, the GUI will open, and if you do, then you will be running the application via the command line for batch processing.

## A.3 Running Analyser

You can run Analyser in two ways depending on the purpose of your usage. One way is to open the application’s graphical user interface, and the other is to use it through a command line. Batch processing of files, training a linear or quadratic regression model, and other machine learning features are only accessible through a command line, so if that is your intention, you should start the app from a command line. On the other hand, if you want to process only one file and do analysis, filtering, or coincidence group reevaluation, you should open the app visually and use the features provided.

In Section A.3.1, I will explain how to use the GUI. In Section A.3.2, I will explain how to use the command line interface. In Section A.3.3, I will describe how filtering out protons works. In Section A.3.4, I will discuss how training regression models is done. In Section A.3.5, I will talk about using pre-trained regression models for making predictions.

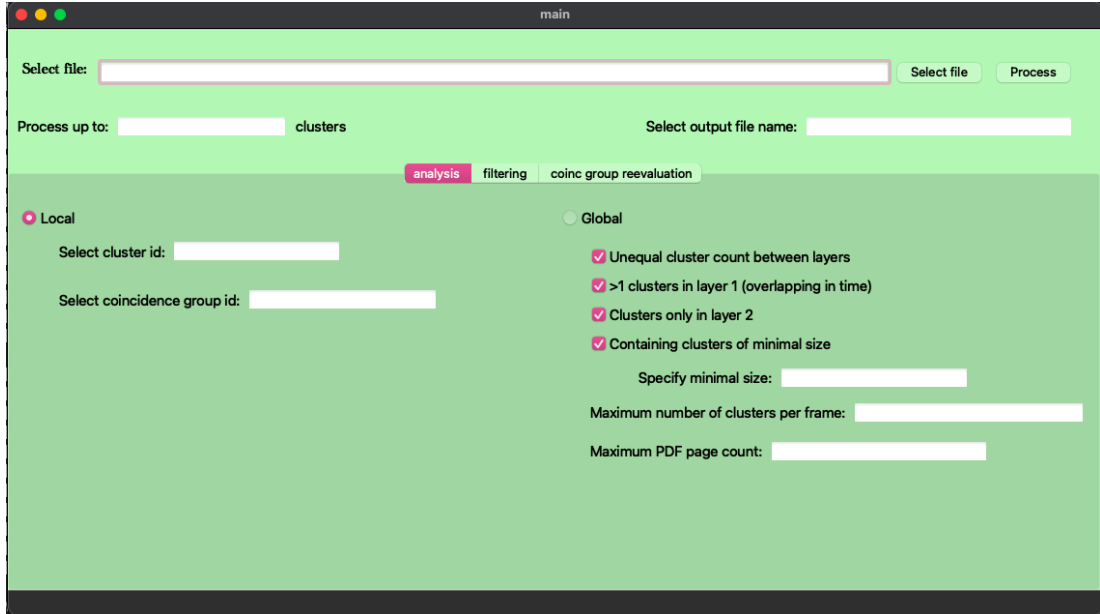
### A.3.1 GUI usage

When opening the app, you will be presented with a window, as seen in Figure A.1. You must select a ROOT file to work with by providing its path. You can click the “Select file” button, and a file dialog will open if you want to select the file without manually typing its path. Then, you can choose how many clusters to look at. When you specify the number of clusters, e.g., 1000, Analyser will work with the first 1000 clusters in the input ROOT file. By not specifying it, the app automatically decides to inspect the whole file. You should also specify the output file name (not including any file extensions). Depending on your further choices, the specified name will be extended with a suffix and an extension (.pdf or .root). This extended name will then be used as the name of the generated output files, which I will explain in the following subsections.

### Analysis

In analysis, there are two main choices you can make: local or global analysis. In local analysis, you can inspect clusters or coincidence groups. When inspecting clusters, you can see an individual cluster regardless of which layer it was detected on. All you need to do is put in the ID of the cluster (a unique integer starting from 0 representing the cluster) you want to inspect. As a result, you will be presented with a visual representation of the cluster and some information about its attributes. The same idea applies to inspecting coincidence groups. You input the ID of a coincidence group (a unique integer starting from 0 representing the coincidence group). Then, you will be able to see a visual representation of the coincidence group, including some information about it, like the number of clusters in layer 1, the number of clusters in layer 2, and its total energy.

In global analysis, you will have three types of output: an interactive view, a PDF file containing a visual representation of each coincidence group the app processes (unless restricted otherwise by setting a maximum page limit), and statistics and histograms that show information about the processed file given in another PDF file and a ROOT file. Based on the name you provided for the output file, the PDF file representing all the coincidence groups visually will



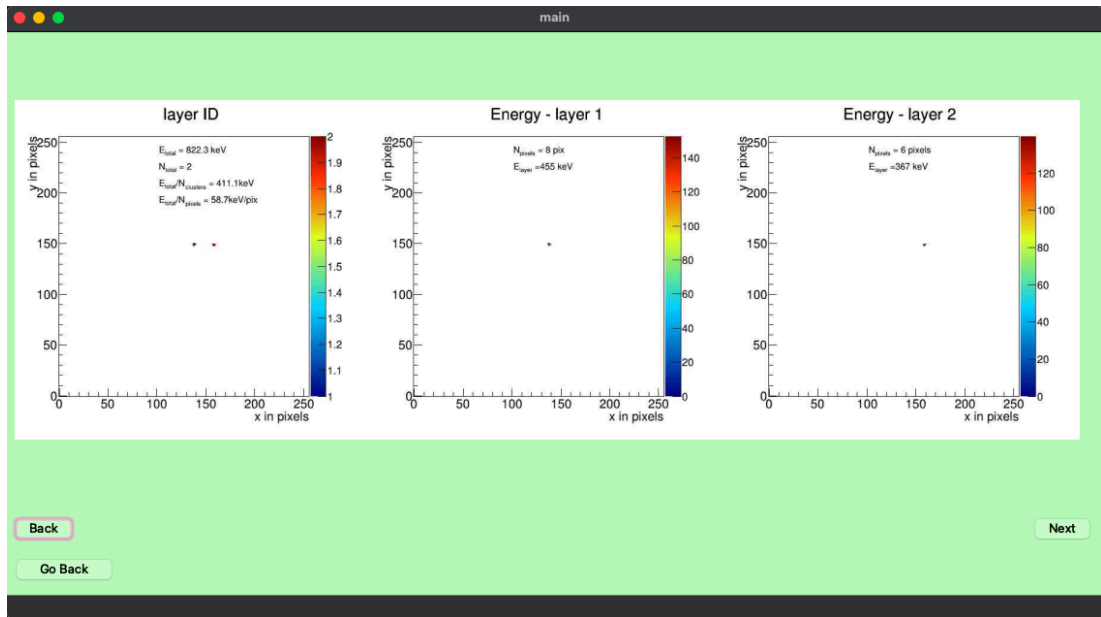
**Figure A.1** The analysis options of the application.

be named “<output-file-name>.pdf”. The PDF and ROOT files representing the statistics and histograms will be named “<output-file-name>\_info.pdf” and “<output-file-name>\_info.root” respectively. All the generated files will be saved under a directory called “generatedResults”, which will be created in the same directory as the input file you provided. These files will all be generated after clicking the “Run” button seen in Figure A.3 (when the processing is over). A message will also appear after the processing is done to let you know that the files were created and everything went well (see Figure A.4). Note that processing large ROOT files can take several minutes.

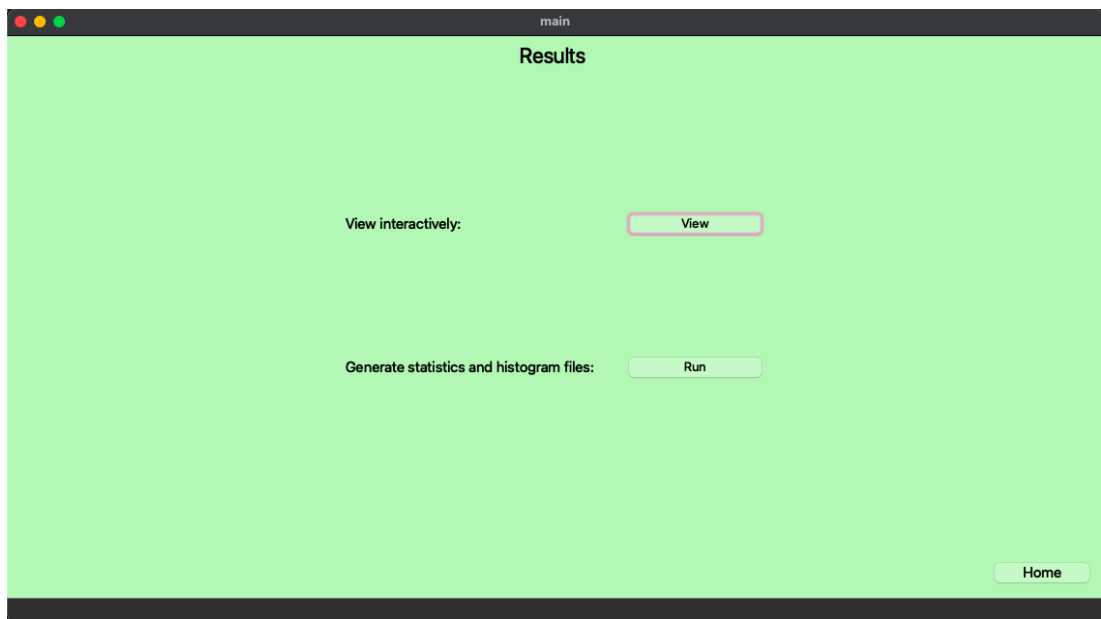
In the interactive view, shown in Figure A.2, you can browse through the coincidence groups that belong to this file by using the “next” and “back” buttons. There are three pixel screens shown. The first one contains all the clusters of the coincidence group regardless of which layer they are. The clusters in layer one will be shown in blue, and the clusters in layer two will be shown in red. The pixel screen also contains information about the total energy of the clusters (represented by  $E_{total}$ ), the number of clusters (represented by  $N_{total}$  or  $N_{clusters}$ ), the average energy per cluster (represented by  $E_{total}/N_{clusters}$ ), and the average energy per pixel (represented by  $E_{total}/N_{pixels}$ , where  $N_{pixels}$  represents the total number of pixels).

The second pixel screen contains information about clusters of the corresponding coincidence group appearing in layer 1. Each pixel is colored according to its energy level (seen on the color bar on the right of the pixel screen). The pixel screen contains information about the total number of pixels (represented by  $N_{pixels}$ ) and the total energy of the clusters in this layer (represented by  $E_{layer}$ ). The same idea applies to the third pixel screen on the right. The only difference is that the third pixel screen contains only clusters appearing in layer 2.

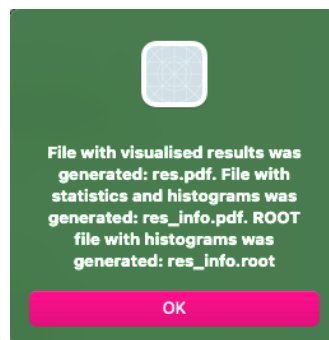
The two additional files that will be generated, containing statistics and histograms (a PDF file and a ROOT file), represent more or less the same information. The ROOT file is more straightforward for developers to work with.



**Figure A.2** Interactive view of a coincidence group.

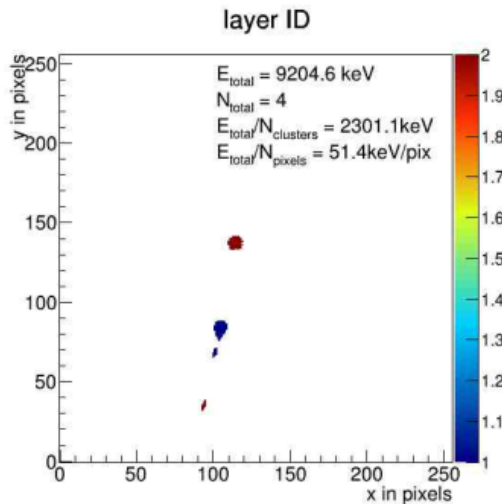


**Figure A.3** Options to choose during the global analysis of a file.



**Figure A.4** Message informing you that the global analysis is finished and the output files are generated.

You can select what statistics you want to compute, and that is precisely what will be presented in the statistics file. Such statistics will include the number of coincidence groups with unequal cluster count in the two layers, with more than one cluster in layer 1, with clusters only in layer 2, or with cluster sizes above a minimal specified size (you then determine what size to be considered minimal and then only clusters of size bigger than that will be accounted for). Figure A.5 gives an example of a coincidence group containing a large cluster that was filtered in based on the cluster's size being bigger than the specified minimal size, 25 pixels.



**Figure A.5** An example of a filtered coincidence group containing a cluster of size bigger than the minimal size specified (25 pixels).

Along with statistics, two two-dimensional histograms will be saved, one representing cluster count and one representing cluster sizes in layers 1 and 2. You can see examples of such histograms in Figure 5.8b. In both histograms, the  $x$ -axis represents layer 1, and the  $y$ -axis represents layer 2.

In the first histogram, cluster count per coincidence group is recorded in both layers. On the right, we have a color bar representing the number of hits for each cell. We can observe that the cell with the highest number of hits (with the color yellow) is cell (1,1), and the second and third highest number of hits are observed in cells (0,1) and (1,0), respectively. That tells us that we had 1 cluster per layer in most coincidence groups present in the input file. Looking at the high number of hits in cells (0,1) and (1,0), we can also see that many coincidence groups had only one cluster regardless of which layer it was in. These insights can also be confirmed by looking at the means of  $x$  and  $y$ , which are just below 1.

The second histogram records the average cluster size for each coincidence group in layers 1 and 2. On the right, we have a color bar representing the number of hits for each cell. We can observe that the cell with the highest number of hits (shown in yellow) is around (11,11), and then there are also cells (0,11) and (11,0) which have a relatively high hit count. By this, we can infer that most clusters had size 11. By referring to the first histogram, where it was shown that a lot of coincidence groups had only one cluster, the high number of hits in cells (0,11) and (11,0) also makes sense, keeping in mind that the most common cluster size

was 11 and the cluster only appeared in one layer. Looking at the histogram, we can also see that most of the cells have recorded hits, although in significantly lower numbers (shown in purple). This tells us that the average size of clusters can vary a lot because there can often be outliers (clusters that appear differently than expected). We see around the edges, though, that some white cells appear with no recorded hits, so as the size gets bigger (towards 50 and above), it gets much rarer to see clusters of that size. This considerable diversity in size is also expressed by the standard deviation, which is much higher than the standard deviation of the first histogram.

Then, there are four one-dimensional histograms regarding the angles  $\phi$  (the azimuth angle representing the direction in 2D space) and  $\theta$  (the angle representing the deviation from the normal to the detector screen). Examples of such histograms are seen in Figures 6.3 and 5.12.

In Figure 6.3, in each histogram, the value of  $\phi$  is given in the  $x$ -axis, and the number of counts for each bin is given in the  $y$ -axis. The first histogram records the value of  $\phi$  in the provided ROOT file. It can be seen that the values are restricted between 0 and 90, which limits the possible azimuth direction of a cluster. The second histogram shows the estimated values of  $\phi$  from Analyser. This estimation is done using the “line of best fit” method. Now, we can see a high number of counts around -3, which makes sense after looking at the ROOT input file manually, as you will notice most clusters tend to lean just a little toward the bottom. This is something that the first histogram cannot tell us.

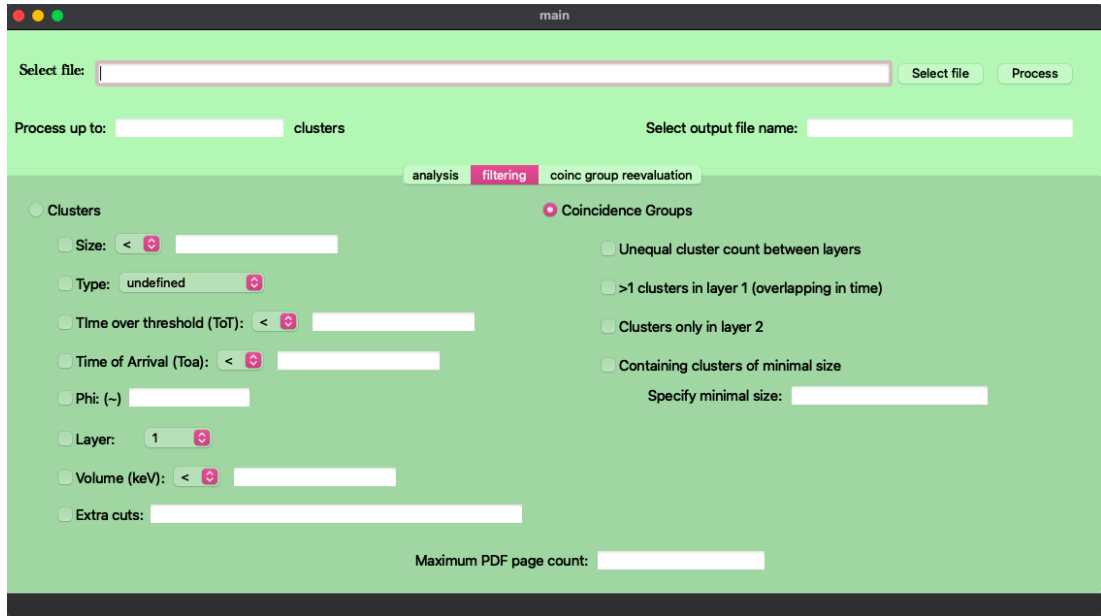
In Figure 5.12, there are experiments done in three different ROOT files, where two histograms are produced for each, one recording the value of  $\theta$  from the file, and the second one recording estimated values of  $\theta$  for coincidence groups with one-to-one matches in clusters in layer 1 and 2. In each histogram, the value of  $\theta$  is given in the  $x$ -axis, and the number of counts for each bin is given in the  $y$ -axis. Looking at the mean presented in each histogram, we can see that the mean of the histogram with estimated values matches very closely with the actual value of  $\theta$  for the given file. In contrast, the mean of histograms with the original values  $\theta$  tends to differ quite a lot from the actual value.

At last, four more histograms will represent the average energy of clusters for each region of the pixel screen defined by the neutron converters between the layers (see Figures 5.9a, 5.9b, 5.9c, and 5.9d). In each histogram, the energy is given in the  $x$ -axis, and the number of counts for each bin is given in the  $y$ -axis. For each region presented, we can observe that most of the clusters have energy below 200 keV, although the standard deviation is very high, so the range for energy is quite large.

## Filtering





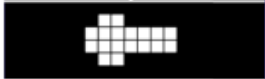

Filtering, using options shown in Figure A.6, can be done according to the properties of clusters or coincidence groups. When you filter by the properties of clusters, there are some main attributes you can filter by, such as:

- size,
  - greater than,
  - smaller than, or



**Figure A.6** The filtering options available.

- equal to
- an integer value representing size in pixels;
- type,
  - undefined,
  - dot,
  - small blob,
  - curly track,
  - heavy blob,
  - heavy track,
  - straight track,
- which is already determined in the source ROOT file (examples can be seen in Figure A.7);
- time over threshold,
  - greater than,
  - smaller than, or
  - equal to
- an integer representing the time over threshold in nanoseconds (will evaluate to true if it holds for any of the pixels of the cluster);
- time of arrival,
  - greater than,

1) Dot		Low energy photons and electrons
2) Small blob		X-ray photons and electrons
3) Curly Track		Gamma rays and electrons (MeV)
4) Heavy Blob		Heavy ionising particles with short range (alpha particles, protons, ...)
5) Heavy Track		Heavy ionising particles (protons, ions, ...)
6) Straight Track		Energetic light charged particles (pions, muons, ...)

**Figure A.7** The cluster types available.

- smaller than, or
  - equal to
- an integer representing the time of arrival in nanoseconds (will evaluate to true if it holds for any of the pixels of the cluster);
- $\phi$  (phi) (in degrees),
    - approximately ( $\pm 5$  degrees) equal to

an integer representing the value of  $\phi$  in degrees;
  - layer,
    - 1 or
    - 2;
  - volume,
    - greater than,
    - smaller than, or
    - equal to

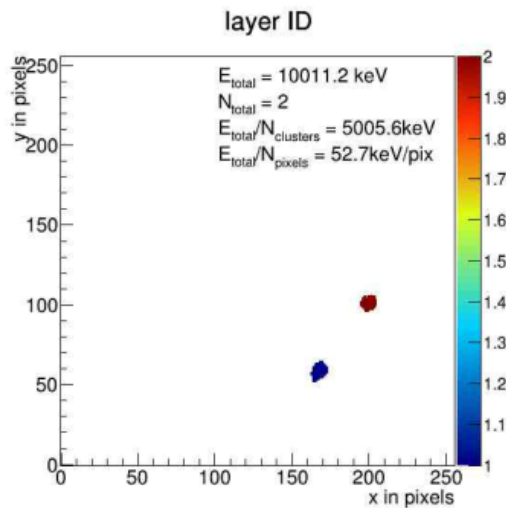
an integer representing the volume in keV;
  - extra cuts,
    - a string containing a valid cut to filter by (see below for description).

However, sometimes these may not be enough, so if you want to filter by some attribute that you know to be in the dataset but is omitted here, or if you want to include more advanced mathematical functions built-in in ROOT (such as `sqrt()`, `sin()`, `abs()`, and so on) on the attributes already available, you can use the “extra cuts” option where you specify a filtering condition. A cut is typically a string

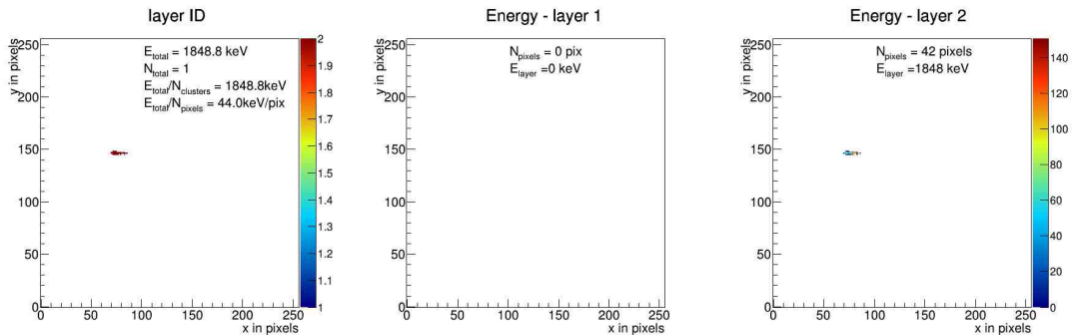


representing the condition to be applied to the data in the ROOT tree. You can combine multiple conditions using logical operators like  $\&\&$  (logical AND),  $\|\|$  (logical OR), and  $!$  (logical NOT). You can use parenthesis and ROOT built-in mathematical functions, and the logical operators follow the usual precedence. E.g., “ $\text{clstrRoundness} > 0.6 \ \&\& \ \text{clstrType} > 3$ ” is an example of a cut, meaning that the roundness of the cluster (attribute  $\text{clstrRoundness}$ ) should be greater than 0.6 and the cluster type (attribute  $\text{clstrType}$ ) should be greater than 3, which means that the cluster should be a heavy blob, heavy track, or straight track. An example satisfying such a cut would be the coincidence group seen in Figure A.8.

When filtering coincidence groups, you can choose filters like unequal cluster count in layers 1 and 2, having more than 1 cluster in layer 1 (so you get clusters overlapping in time), only appearing in layer 2, or having clusters with size bigger than the minimal size given in pixels (if a coincidence group contains at least one cluster of size bigger than the specified minimal size), which again remains to be determined by you. The filters are disjunctive, so whenever any of the filtering conditions are fulfilled, the coincidence group is considered. An example of filtering coincidence groups by the attribute of having only clusters in layer two is seen in Figure A.9.

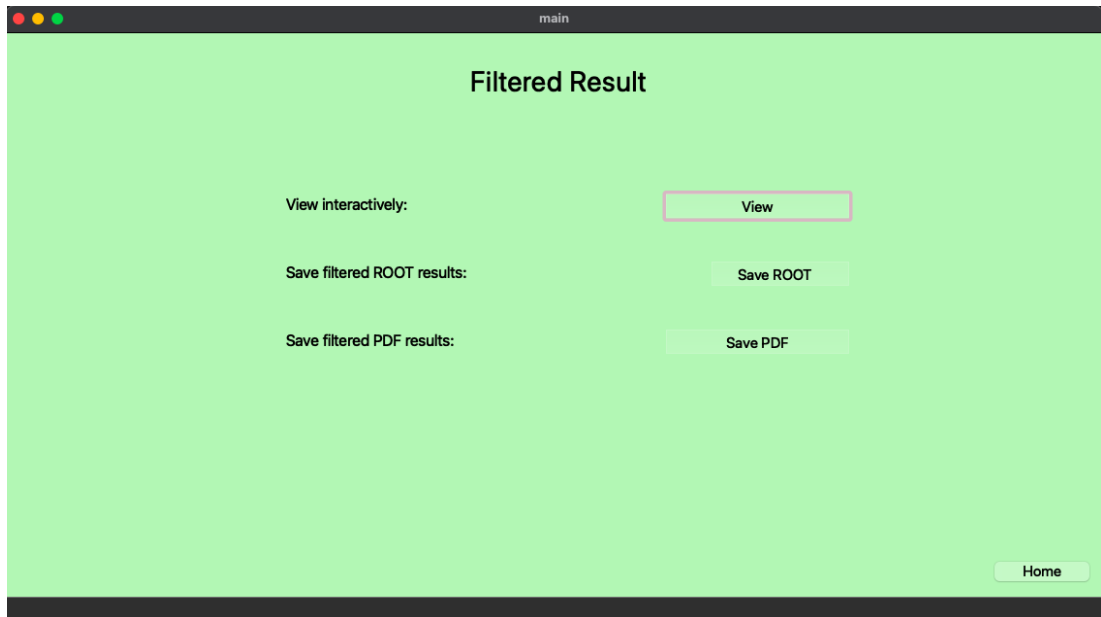


**Figure A.8** An example of a coincidence group with clusters satisfying the cut “ $\text{clstrRoundness} > 0.6 \ \&\& \ \text{clstrType} > 3$ ” during filtering of clusters.

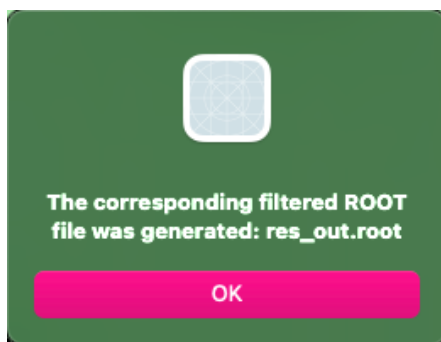


**Figure A.9** An example of a filtered coincidence group with clusters only in layer 2.

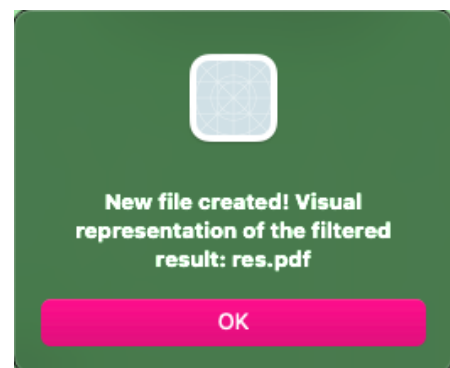
In the output, similarly to analysis, you will get an interactive view that you can browse through, a PDF file of the filtered data (when clicking the “Save PDF” button seen in Figure A.10), and a ROOT file containing only the filtered clusters/coincidence groups (when clicking the “Save ROOT” button seen in Figure A.10). The PDF and the ROOT file will be named “<output-file-name>.pdf” and “<output-file-name>\_out.root” respectively. These files will be generated only after the filtering is done and a message reporting that is displayed to you (see Figure A.11). They will also be found under a directory called “generatedResults” in the same directory as the input file provided.



**Figure A.10** Possibilities for obtaining filtering clusters or coincidence groups.



(a)

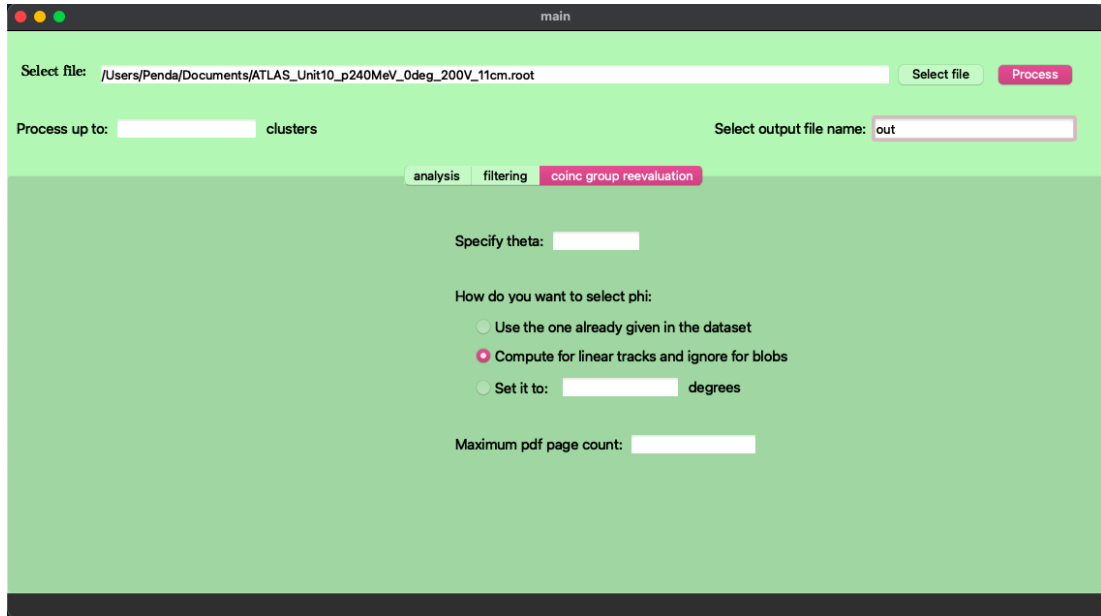


(b)

**Figure A.11** Messages informing you that the filtered ROOT and PDF files have been generated.

### Coincidence Group Reevaluation

This feature, shown in Figure A.12, allows for reevaluating coincidence groups based on spatial information. You can specify the value of angle  $\theta$  (theta) in



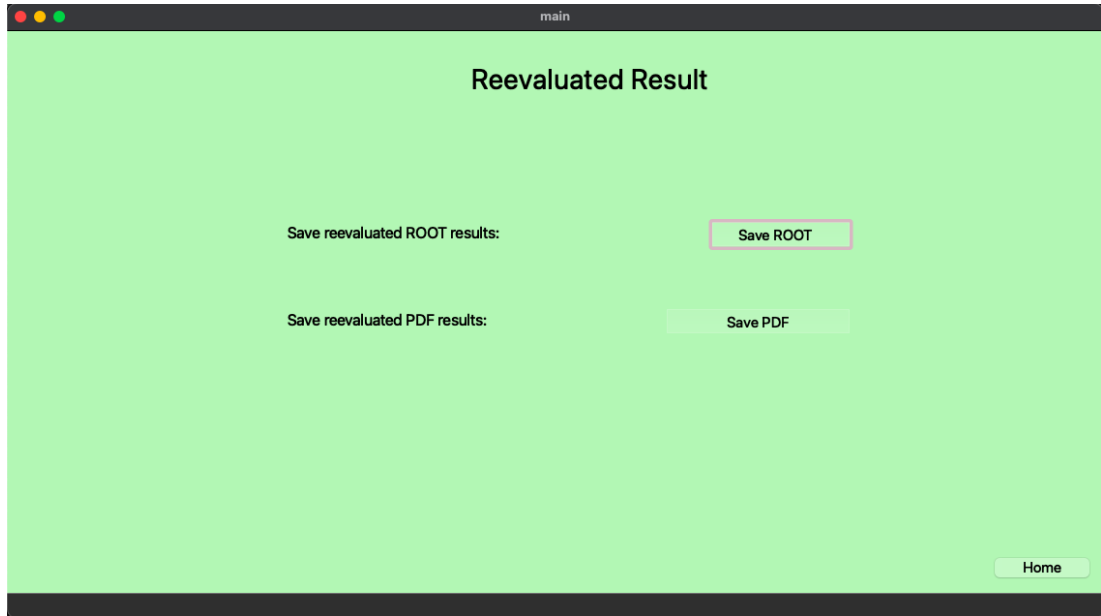
**Figure A.12** The options for coincidence group reevaluation.

degrees (the deviation of the stream of particles from the normal to the surface of the detector), as it can be used to determine the major and minor of the ellipses Analyser will use for reevaluation. The particles are hitting the detector at an angle  $\theta$  from the normal to the surface of the detector. Specifying  $\theta$  enables one to check the position of clusters in both layers corresponding to the same particle. Analyser can then partition clusters into groups representing different particles if the clusters appear too far from each other in space. This represents an additional restriction on top of temporal information for creating coincidence groups.

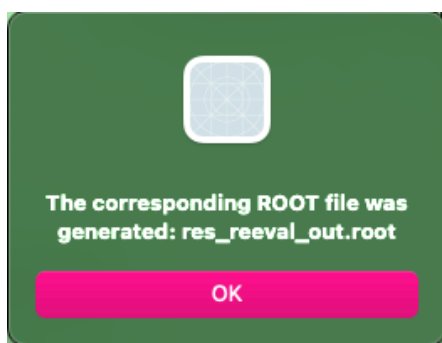
Then, you have three options for determining the angle  $\phi$  (phi). You can choose to use the  $\phi$  already computed in the dataset, compute it on the go for linear tracks using the line of best fit estimation and ignore it for blobs (as the  $\phi$  direction is not important for round clusters), or set it to a certain value in degrees that you think is appropriate and general enough to represent the dataset well and not have a bad effect on the results.

Examples of reevaluated coincidence groups are shown in Figures 5.13 and 5.14. Two proton particles that happened to appear at the same time are reevaluated using spatial information and separated into two different coincidence groups.

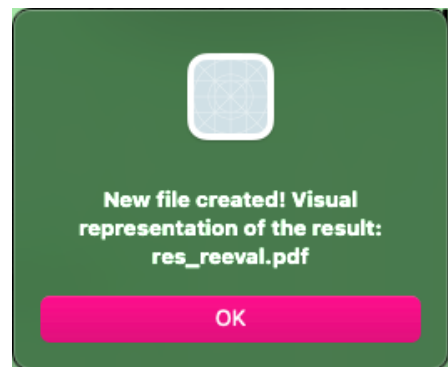
When reevaluating a ROOT file and assigning new coincidence groups, you can choose to save the result as a reevaluated ROOT file (when clicking the “Save ROOT” button seen in Figure A.13), a reevaluated PDF file (when clicking the “Save PDF” button seen in Figure A.13), or both. The reevaluated ROOT file will contain all the clusters of the original file but with new coincidence groups assigned based on the computations done during reevaluation. It will be named “<output-file-name>\_reeval\_out.root”. The reevaluated PDF file will contain all the reevaluated coincidence groups unless restricted by a maximum PDF page limit that you can specify (strongly recommended so that you do not end up with a PDF file containing hundreds of millions of pages). This file will be named “<output-file-name>\_reeval.pdf”. These files will be generated only after the reevaluation is done and a message reporting that is displayed to you (see Figure



**Figure A.13** Output possibilities for reevaluating coincidence groups.



(a)



(b)

**Figure A.14** Messages informing you that the reevaluated ROOT and PDF files have been generated.

A.14).

### A.3.2 Command Line Usage

Using the command line, you can do batch processing for global analysis, filtering clusters, training regression models, using pre-trained models to make predictions, and filtering protons out of ROOT files. To get the desired results, you can specify certain arguments through the command line. The provided arguments are:

- `-h, --help` – used to display what parameters you can specify and what each of them means,
- `-f, --files` – used to list the paths to the ROOT files (separated by commas) that you want to use as datasets in batch processing,

- `--ga`, `--globalanalysis` – used to specify that you want to perform global analysis on the data,
- `--fc`, `--filterclusters` – used to specify that you want to filter clusters from the data,
- `--fp`, `--filterprotons` – used to specify that you want to filter protons out from the data (i.e., the output will contain clusters that are likely not protons),
- `-t`, `--train` – used to generate a regression model with SGD (stochastic gradient descent) where the files specified by `--files` are used as datasets to train on using features given by specifying `--features`,
- `--fe`, `--features` – features to use for the regression model separated by commas, where the last one represents the output (y),
- `--pd`, `--predictdataset` – used to specify that you want to predict a given dataset using a model that was previously trained and check the model’s performance,
- `--model` – the path to the model file to be used for prediction,
- `--cut` – a string representing a valid cut to filter clusters by (enclosed in double quotes),
- `--outfilename` – the name of the output file serving a similar function as that described in the GUI mode,
- `--maxPdfSize` – the maximum number of pages a resulting PDF file can contain,
- `-p`, `--testfile` – the path to the ROOT file that will be used to test regression model predictions,
- `--lr`, `--linearreg` – used to specify that linear regression should be used,
- `--qr`, `--quadraticreg` – used to specify that quadratic regression should be used.

There are some restrictions on argument usage so that the program gets all the information necessary to complete a certain task. I will give some notes on how to use the arguments to perform the task you need:

- The five main tasks you can perform through the command line are global analysis, filtering of clusters, filtering protons out, training a regression model, and using a previously trained model to predict a ROOT dataset and check its performance. For this reason exactly one of `--ga`, `--fc`, `--fp`, `-t`, `--pd` needs to be set.
- `--files` should always be specified (unless you are performing prediction and `--pd` is specified) and contain at least one file path because the program needs at least one ROOT dataset to work with.

- If `-t` is specified, you can train models. Still, more information is necessary to continue with the task, so the following arguments are required together:
  - Exactly one of `--lr` and `--qr` needs to be set, indicating the regression type you want.
  - You need to provide at least two valid features to use for training by using the argument `--features`.
- `--outfile` should always be specified in order to know where to save the results (obligatory).
- If you specify one of `--ga`, `--fc`, or `--fp`, then you also specify:
  - `--maxPdfSize` to set a maximum page limit for your resulting PDF file (optional but recommended).
- When you set `--maxPdfSize` to 0, no PDF file will be generated.
- If your goal is global analysis, you use `--ga`, and no additional arguments are required.
- If your goal is filtering clusters, then you use `--fc`. Additionally, you must also set the argument `--cut` and define a valid nonempty cut to filter clusters by. The cut represents the same thing as in the GUI, and you can filter by any of the attributes available in the ROOT file.
- If your goal is filtering protons out, you use `--fp`, and no additional arguments are required. The result will be a ROOT file with clusters that are likely not protons.
- If your goal is predicting a ROOT file using a pre-trained model, you use `--pd` and set the following arguments:
  - You should give exactly one test file to test the model on via the argument `--testfile`.
  - You need to provide the same features you used to train the model by using the argument `--features`.
  - You should give exactly one file path of the pre-trained model file by using the argument `--model`.

The result will be a report of the Root Mean Squared Error in the command line and a CSV file containing the predictions.

If any condition fails to hold, the program will exit and return with an error message indicating the problem.

I will list some example usages for the five main tasks:

- Global analysis

```
./TimePix3Analyser --files path/to/file1,path/to/file2
                   --ga --maxPdfSize 50 --outfile "res"
```

You want to perform global analysis on “file1” and “file2”, and you define the name of the resulting files to be “res” and limit the size of the PDF file generated to 50 pages.

- Filtering of clusters

```
./TimePix3Analyser --files path/to/file1,path/to/file2
                    --fc --outfilename "res"
                    --cuts "clstrRoundness>0.6&&clstrType<3"
```

You want to look at files “file1” and “file2” and filter them by the roundness of clusters (attribute *clstrRoundness*) being bigger than 0.6 and the cluster type (attribute *clstrType*) being smaller than 3, which means only clusters that are dots, small blobs, or curly tracks. You further define the name of the resulting file as “res”.

- Filtering protons out

```
./TimePix3Analyser --files path/to/file1,path/to/file2
                    --fp --maxPdfSize 0 --outfilename "res"
```

You want to look at files “file1” and “file2” and filter protons out of them. You further define the name of the resulting file as “res” and by setting the maximum PDF size to 0, you do not want any PDF file to be generated as a result.

- Training a regression model

```
./TimePix3Analyser --files path/to/file1,path/to/file2
                    --features clstrLength,clstrRoundness,clstrType
                    --qr --train --outfilename model
```

You want to train a quadratic regression model using the files “file1” and “file2” as training data. You use features *clstrLength* and *clstrRoundness* as independent variables in training and feature *clstrType* as the dependent one. You further define the name of the resulting file as “model”.

- Predicting a ROOT file using a pre-trained model

```
./TimePix3Analyser --testfile path/to/file1
                    --features clstrLength,clstrRoundness,clstrType
                    --pd --model path/to/model --outfilename preds
```

You want to test the performance of your pre-trained model on a test ROOT file called “file1”. You get your serialized model from the file “model” and set the same features you used to train it. You also specify that the output CSV file with predictions should be saved as “preds.csv”.

### A.3.3 Filtering Protons Out

When filtering protons out from a ROOT dataset (performed via command line), you should follow the steps described in Section A.3.2 for running the application. When the processing is done, there are two files generated. One of them is a PDF file containing visual representations of the filtered result. This will be called “<output-file-name>\_noprotons.pdf”. However, if the maximum PDF page count is set to 0, this PDF file will not be generated at all. The other file generated is a ROOT file containing the filtered clusters (without proton clusters that were removed in the process). This will be called “<output-file-name>\_noprotons.root”. Both of these files will be found in the directory called “generatedResults” created in the same directory as the first input file you provided.

### A.3.4 Training Regression Models

When you want to train a regression model on the ROOT datasets you provided via the command line, you should follow the steps described in Section A.3.2 for running the application. During processing, the input datasets provided will be fed into the multivariate linear or quadratic regression model one by one until there are no more clusters to process. When the model is trained, the computed coefficients will be printed out for you, and the model will be saved in a file called “L\_<output-file-name>.dat” if you performed linear regression, or “Q\_<output-file-name>.dat” if you performed quadratic regression. This generated file will contain a serialization of the model you just trained. The file can later be used for predictions.

### A.3.5 Using a Pre-trained Model for Predictions

When you want to use a pre-trained model to test its performance and make predictions, you should follow the steps described in Section A.3.2 for running the application. Analyser will deserialize the given model you provide using the command line via the argument `model`. It will use the test ROOT file you provided to make predictions on it for testing. The program will compare these predictions to the real ones provided in the datasets and estimate how good the performance of this model is by computing the root mean squared error. Ultimately, it will present this error to the user in the command line so the user can evaluate how well this model performed.

To present the predictions to you more clearly, it will also save them into a CSV file. The CSV file will contain three columns: the cluster index, the real output value, and the predicted output value. The name of the CSV file will be “<output-file-name>.csv”. It will be saved under a directory called “generatedResults”, which will be created in the same directory as the first input file you provided. A snippet of such a CSV file after using a previously trained model to predict the value of  $\theta$  is shown in Figure A.15.



1	Cluster ID	Real theta	Predicted theta
19	17	12.4074	14.6117
20	18	13.8186	10.7189
21	19	13.8186	10.7189
22	20	13.8186	10.7189
23	21	13.8186	7.87303
24	22	13.8186	7.87303
25	23	13.8186	10.7189
26	24	12.4074	14.6117
27	25	17.2823	4.51634
28	26	13.8186	7.87303
29	27	13.8186	7.87303
30	28	19.1803	18.5724

**Figure A.15** A snippet from a CSV file containing model predictions.

# B Attachments

In the electronic attachment, the following information is included:

## B.1 Analyser - Source Code

This folder includes all the source code used to build all of the tools mentioned as part of this thesis. It contains C++ and Cmake files.

## B.2 Analyser - Executables

This folder will contain two folders inside with executables for MacOS and Linux operating systems.

## B.3 Analyser - User Documentation

This folder will contain the user documentation for the application that contains installation instructions, features, and tutorials on how to run the application.

## B.4 Link to the Test Data

This folder (the dataset drive) contains some ROOT files that can be used to test the application. There are 5 files with proton data and one file with neutron data. Note: the size of most files is in gigabytes.

## B.5 Link to GitLab Repository

The repository of the project (where the source files are) can be found at <https://gitlab.mff.cuni.cz/smajljap/timepix3gui>.