**Charles University**

**Faculty of Mathematics and Physics**

**BACHELOR THESIS**

**2021 Tomáš Čelko**



**BACHELOR THESIS**

Tomáš Čelko

**Support for annotating and classifying particles detected by TimePix3**

Department of Software and Computer Science Education

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

Study programme: Computer Science

Specialization: IOI

Prague 2021

[Sample: Bound Sheet – a signed copy of the "master thesis assignment". **This assignment is NOT a part of the electronic version of the thesis. DO NOT SCAN**.]

I declare that I carried out this bachelor thesis independently 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 Coll., 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 paragraph 1 of the Copyright Act.

In…...... date............ signature

[Sample: Here you may thank whoever you wish (the supervisor of the thesis, the consultant, the person who lent the software, literature, etc.)]

Title: **Support for annotating and classifying particles detected by TimePix3**

Author: Tomáš Čelko

Department / Institute: Department of Software and Computer Science Education Supervisor of the master thesis: First and last name with titles, workplace

Abstract:

Keywords: Cluster classification Timepix3

Contents

[Introduction / Preface 1](#_Toc70708499)

[1 Analysis and work on the project 4](#_Toc70708500)

[1.1 The Medipix family and TimePix3 detector 4](#_Toc70708501)

[1.2 Input format and calibration 5](#_Toc70708502)

[1.3 Calculating the features 6](#_Toc70708503)

[1.4 Cluster viewing 7](#_Toc70708504)

[1.5 Description Generator 10](#_Toc70708505)

[1.6 Classifier 10](#_Toc70708506)

[2 Development documentation 11](#_Toc70708507)

[2.1 Important objects 11](#_Toc70708508)

[2.2 Viewer 11](#_Toc70708509)

[2.3 Filter 12](#_Toc70708510)

[2.4 Description Generator 13](#_Toc70708511)

[2.5 Classifier 13](#_Toc70708512)

[3 Experiment 14](#_Toc70708513)

[3.1 Baseline single-layered Model 14](#_Toc70708514)

[3.2 K-Fold cross validation 14](#_Toc70708515)

[3.3 Multi-layered classifier 14](#_Toc70708516)

# Introduction / Preface

In the field of nuclear physics, there have been many efforts to visualize and detect elementary particles. For this purpose, various detectors were invented, which are now members of the so-called Medipix detector family. The most recent member of the family is called the TimePix3 detector. In terms of elementary particle detection, TimePix3 achieves state-of-art performance. During a specified timeframe, the detector captures a set of particle trajectories. However, in some cases, a single particle emits the secondary particles that can interact. Instead of detecting the particles one by one, we analyze groups of such particles - so-called clusters.

So far, there have not been many publications released in the processing, filtering, visualization, or classification of these clusters. In the filtering task, one needs to make sure the algorithm is fast because the size of the cluster dataset captured by TimePix3 detector can reach gigabytes of data over a short timeframe. The classification process can be challenging because the trajectory of the cluster depends on the angle at which the particle enters the field of the detector. Furthermore, the distribution of various types of clusters in the standard observation is usually very uneven. For instance, most of the data received from detectors like ATLAS [link] contains only simple traces, which consist of a few pixels and does not provide much information for the analysis. This fact causes problems for many machine-learning-based approaches because the datasets of the rare, more complicated clusters often have a minimal size.

**Goals of the thesis**

Our main goal is to create a set of processing tools that would enable physicists to analyze the clusters and their properties. Firstly we need to provide support for filtering the clusters based on their attributes. Secondly, we visualize them individually so that the users can see the cluster as a 2D and 3D image. Because the number of clusters in some datasets can be overwhelming, another goal is to make a tool, which could calculate the properties for the whole collection of clusters. The calculated properties of the clusters are then used to create a neural network-based classifier capable of classifying various clusters, which is our final goal. Eventually, the developed classifier could also select extraordinary clusters, displaying exotic or even unseen particles.

**Thesis layout**

**Related work**

* Detecting elementary particles with Timepix3 detector [4]

[Sample: The actual text of the master thesis organized hierarchically into chapters and subchapters, each chapter always on a new page. It is advisable to use formatting of chapters.

The range of the thesis is given in standard pages. One standard page contains 30 lines of 65 characters (the line 30 is for the page number).

The font recommended is twelve point (12 pt) with a standard distance between lines (line spacing 1.5). The text of mathematical theorems is usually printed to highlight the so-called slanted font, which is similar to italics. Text is written in blocks. A new paragraph is usually separated by indentation of the first row.

It is primarily recommended to use single-sided printing, though two-sided printing is not prohibited. For two-sided printing it necessary to consider the correct edge width. The reverse of the title page remains blank.

Top, bottom and right margin 25 mm, left margin 40 mm. The entire text must keep the same layout. The work is printed on white A4 paper.

Pictures, diagrams and tables are numbered so that they can be referred to in the text. They must bear a description usual for scientific papers. Descriptions of tables, figures and diagrams, including their numbering are given below, with the same size font as the text of the thesis, and below the graphical representation the source is stated in italics and font size is smaller than the basic text.

Abbreviations used in the text should always be explained at the first occurrence (in parentheses, or a footnote, if it is a more complicated explanation of the term or abbreviation). At the same time a list of abbreviations, including their explanation is given.

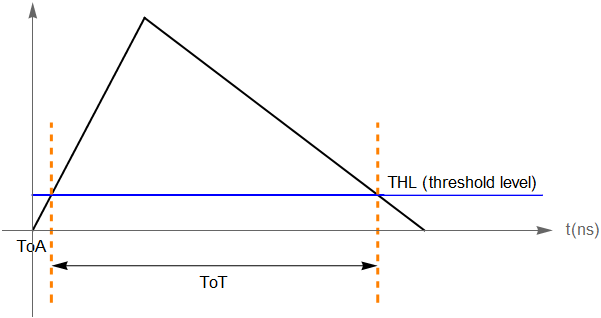
A longer text by another author must be given in quotation marks, or otherwise indicated, and properly cited.]

# Analysis

## The Medipix family and the TimePix3 detector

It was the 1990's when some of the researchers from CERN came with an idea to transfer the devices, primarily developed for experiments in LHC beyond particle physics. The first collaboration with such a goal started under the name Medipix1. It was the collaboration of the University of Freiburg, University of Glasgow, and Napoli and Pisa Universities together with CERN. So far, there have been four Medipix collaborations, each with its own specific goals. The chips developed in these collaborations are known as the members of the Medipix detector family.

The first member of the family was the Medipix1 chip (1997), consisting of 64 x 64 pixels acting similarly to an electronic camera – counting the hits of elementary particles while the shutter is open. A few years later, the Medipix2 chip was developed, leading to the first Timepix chip (2006) being invented. This was the first chip that can be programmed to record one of the following properties:

* **Particle hit count** (similar to Medipix1)
* **Time over the threshold** – Each pixel is assigned an energy threshold level. When a charged particle approaches the pixel, the energy captured by the pixel rises. The time interval while the energy remains above the threshold we call the time over the threshold (ToT). This attribute is often measured as the number of ticks of the detector clock.
*  **Time of the arrival** – The absolute time since the start of measurement until the energy threshold level is reached we call the time of arrival (ToA).

In 2013 the new TimePix3 chip was introduced to the family. "Timepix3 is a general-purpose integrated circuit suitable for readout of both semiconductor detectors and gas-filled detectors. Compared to its predecessor Timepix the circuit has more functionality, better time resolution, and more advanced architecture for continuous sparse data readout with zero-suppression." [1] Zero-suppression means that there is no data output from the detector unless nonzero energy input is captured. This allows better efficiency in data collecting and storage but also in data analysis. The device utilizes a 256x256 pixel matrix where the size of each pixel is 55μm, achieving a time resolution of 1.56ns. The TimePix3 chip is nowadays used in the CERN LHC to detect sets of elementary particles, so-called clusters.

## Input format and calibration

Because the output of the TimePix3 chip over some timeframe can contain multiple clusters, we use the Clusterer application [4] to process the raw input from the detector and separate the clusters from each other. This means that instead of processing the data in the raw format, we use the data in the MM (clustered) format as shown in the chart.

|  |  |  |  |
| --- | --- | --- | --- |
| **FILE** | **DESCRIPTION** | **FORMAT** | **EXAMPLE** |
| ini | Initial file, contains references to the specific .cl and .px files | [Measurement] PxFile=[Relative path from the parent direcotry of .ini to px file]  ClFile=[Relative path from the parent direcotry of .ini to cl file] | Measurement 123 PxFile= Clusters\_px.txt ClFile= Clusters\_cl.txt |
| cl | File where each line represents a single cluster and a reference to its pixels in px file | [First ToA (float - 4B)] [Pixel Hit Count (unsigned integer – 4B)] [LineOfStart in px file (unsigned integer)] [Byte of start in px file (unsigned integer)] | 12345.647 100 5 30 |
| px | File with all of the pixels of the clusters in cl file, separated by the hash tag "#" symbol | [x coordinate of the pixel(unsigned byte - 1B)] [y coordinate of the pixel (unsigned byte - 1B)] [ToA (float - 4B)] [Energy in keV (float - 4B)] | 123 128 15540 14.235 |

There are two kinds of MM formats, which only differ in the px file:

* ***Calibrated*** data have the structure as displayed in the chart []
* ***Non-calibrated*** data are very similar to the calibrated ones, but instead of the energy attribute, they use the *ToT* attribute

To find out whether a file is calibrated or not, we can open the px file and look for the last column. If the decimal part of the values is zero in every pixel, we know we are dealing with non-calibrated data because the *ToT* is measured as a number of ticks of the detector, which is an integral value. However, if the values have a non-trivial decimal part that indicates the data is already calibrated.

The calibration is the process of replacing the *ToT* attribute with the corresponding energy. The energy deposited in the pixel is a function of the *ToT* but also four other parameters of a pixel, denoted by the letters "a," "b," "c," and "t." These parameters are set during the measurement and are usually stored in separate text files as a 256x256 matrix of the decimal numbers. It holds:

In the equation, *E* represents the energy deposited in the pixel, with *a*, *b*, *c,* and *t* being the calibration parameters.

## A little about neighbors

Neighboring of the pixels is an essential term when it comes to 2D image analysis. In general, two pixels are called neighbors if there is a relatively small distance between them. To be more specific, there are two known kinds of neighbors:

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |

* ***8-neighbors of a pixel p***

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |

* ***4-neighbors of a pixel p***

And for future use, we will also recognize another kind of neighbors:

* ***Y-neighbors of a pixel p***

This kind of neighbors has two main variants:

Variant AVariant B

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |

For each variant, we also consider their symmetric alternatives to belong to the same variant.

## Calculating the features

The critical point of all our following work is calculating the properties of a cluster, which is the main topic of this subchapter. We will briefly discuss the features and the information they give us about the cluster. For this feature calculation, we created the ClusterCalculator library. The most important class of the library is the AttributeCalculator. When given a set of properties of a cluster, it calculates their values. The properties we calculated range from straightforward to more sophisticated:

* ***Total, average and maximum energy, the standard deviation of energy,*** and ***low energy pixels count***provide information about the energy distribution of pixels in a cluster. The total energy of a cluster is defined as a sum of the energies of each pixel in the cluster. A pixel is considered to have low energy if its energy is less than 10keV.
* ***Pixel count*** attribute reflects the size of a cluster
* ***Width*** and ***convexity*** attributes are both based on the convex hull of a cluster. Let denote the vertices of the convex hull of a cluster. Then the convexity is defined as follows: .

The convexity of a cluster provides us the information about its shape because the more complicated clusters usually tend to have a concave shape. In contrast, the simple ones are often more convex. Width is defined as where *d* represents the distance between given vertices. For the function *d,* we chose to use the Euclidian distance in a plane: where

* ***The standard deviation of the ToA***attribute captures the information about the timespan of a cluster.
* ***Vertex count, Crosspoint count,*** and ***Branch Count*** reflect the possible number of particles in a cluster and the shapes of the trajectories in the cluster. A pixel is considered to be a vertex if it only has one neighboring pixel. Crosspoints are the pixels where the trajectories of the different particles meet. These particles we find as the ones with three or more 4-neighbors or 3 Y-neighbours (both variants of Y-neighbours are possible). Crosspoints of a set are denoted by A branch found at the *k*th iteration is a set of pixels defined as follows: Let *P* denote a set of pixels in a cluster *C*. Let denote a graph such that there exists a bijection *b* between *P* and *V*. Furthermore, for each pair of 8 - neighboring pixels in the cluster there is a corresponding edge in E*.* Then we say P is a branch if and only if its corresponding graph G is a path, P has the maximum possible size, and at least one of the following conditions holds:
  + k = 0
  + none of the pixels in P are part of any of the previousbranches in C

## Cluster viewing

In this subchapter, we will discuss all the features of the application for cluster visualization – ClusterViewer. This application contains tools to display and analyze the properties of a cluster. The features range from a simple 2D image to a 3D image and the analysis of the particle class.

*2D view*

To get a better overview of the cluster dataset, we decided to make the Cluster Viewer application. The primary objective of the viewer is to visualize the cluster as a bitmap, assigning the colors based on the energy of the pixels. This 2D image is represented as a 256x256 bitmap. In the cases where the same pixel is hit multiple times in a cluster, we decided to display the one with the highest deposited energy. Because the low-energy pixels prevail in most clusters, we chose to map energies to color space logarithmically, which seemed to distinguish the energies without needing a wide range of colors.

*Skeletonization*

Skeletonization of a binary image is defined as the thinning process that outputs a simpler version of the original image, the so-called skeleton. An important requirement for skeletonization is to have the image in binary format – each pixel is either white or black. Because the image of a cluster is not binary, we assign a pixel with nonzero energy a value of 1 and a pixel with zero energy a value of 0. This skeleton should preserve the original shape of an image. Because the definition of skeletonization is not exact, various approaches can be used, each possibly outputting a unique skeleton.

For this purpose, we decided to modify Zhan-Shuen's algorithm for thinning binary digital patterns [6] in the following way:

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
| 1 |  |  | |
| 2. |  |  | |
| 3 |  |  | |
| 4. |  |  | |
| 5. | If condition(S,p) and p.energy<t |  | |
| 6. |  |  | |
| 7. |  |  | |
| 8. |  |  | |
| 9. |  |  | |
| 10. |  |  | |
| 11. |  |  | |
| 12. |  |  | |
| 13. |  |  | |

|  |  |
| --- | --- |
|  | |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

The whole skeletonization process consists of using the twice. First time with value set to 10 keV and then we repeat the process with set to infinity. As a result, the first iteration aims to remove the halo effect while still preserving the shape. The second iteration with acts the same as the original version of Zhan-Shuen's algorithm. A different approach would be to filter the low-energy pixels right away, but that does not guarantee that the cluster remains connected, which is a problem for future branch analysis. Because the thinning process returns non-trivial data for bigger clusters, we optimized the algorithm for those (~70-1000 pixels). The differences in the output of the original algorithm and our modification can be seen in the figures []



Figure 3 - Skeletonized cluster (modified Zhan-Shuen's algorithm)

Figure 2 - Skeletonized cluster (original Zhan-Shuen's algorithm)

Figure 1 - Original 2D image of cluster

*Collection Histogram*

Another feature of the application is the Collection Histogram. That is the histogram of the currently loaded collection of clusters, representing the number of clusters with respect to their PixelCount. This information could provide insight on the distribution of PixelCount property in the currently viewed dataset. 

*Pixel Histogram*

Pixel Histogram works similarly to the Collection Histogram, except for the fact that it will depict the histogram of the pixels in the currently loaded cluster. The default displayed property of the pixel is its energy. This histogram could be helpful when deciding the class of a given cluster, because similar classes tend to have a similar energy distribution.



*3D reconstruction*

To provide the user with a 3D trajectory image, we need to calculate the z-coordinate for each pixel. The z coordinate is a function of the relative time of arrival , where  is the difference between the arrival time of the particular pixel and the minimum time of arrival in the entire cluster. It holds:

 [2]

Parameters  (depletion voltage), (bias voltage),  (mobility), and  (thickness of the sensor) are specified at the beginning of the measurement and remain constant for the whole duration of the measurement. This way, we calculate the z-coordinate; thus, we transform two-dimensional points into three-dimensional, which we can then show visually as a 3D scatter plot. For a better viewing experience, we will add an option to rotate the image around the x and y axes.

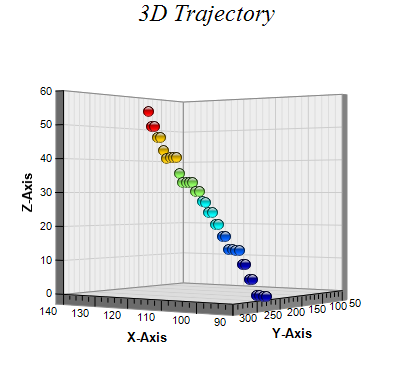


Figure 4 - 3D reconstruction of a track of the particle with a linear trajectory

*Center Finder*

When analyzing the shape of a cluster, it could be useful to know where the center of the event is. The center is usually the point with high energy located at the intersection of the visible trajectories (if there are any). If we manage to find this point correctly, we could then start analyzing each trajectory starting from the core pixel of a cluster event. However, this task proved quite challenging, especially for the complicated clusters, because there can be multiple points with the high energy level lying at the intersection of the trajectories. For this task, we proposed an algorithm based on the energy of the pixel and its surrounding pixels: Let C denote an arbitrary set of pixels and let  be the set of pixels surrounding the pixel (possibly where can be the Euclidian distance [] and ). Then, we define the weighted surrounding energy of a pixel with the weight as with being the energy of a neighboring pixel . For each pixel, the surrounding energy is used to calculate the center cost function.

Let represent the set of pixels of a cluster and let denote its skeletonized version. The center cost function is defined as follows:

In our algorithm, we set . Then the center pixel of a set of pixels is computed simply as the

*Branch Analyzer*

One of the more sophisticated features of the ClusterViewer application is the BranchAnalyzer. Its main task is to analyze the given cluster and search for the possible trajectories of various particles in the cluster. These trajectories - branches are then distinguished by their colors. Each branch can also have its subbranches. We say the branch  is a subbranch of the branch , if it starts in one of the crosspoints of branch b. To analyze the branches, we proposed the following algorithm:

|  |  |  |
| --- | --- | --- |
|  | | |
| 1. |  |  |
| 2. |  |  |
| 3. |  |  |
| 4. |  |  |
| 5. |  |  |
| 6. |  |  |
| 7 |  |  |
| 8. |  |  |
| 9. |  |  |
| 10. |  |  |
| 11. |  |  |
| 12. |  |  |
| 13. |  |  |
| 14. |  |  |

Table 1

|  |  |  |
| --- | --- | --- |
|  | | |
| 0. |  |  |
| 1. |  |  |
| 2. |  |  |
| 3. |  |  |
| 4. |  |  |
| 5. |  |  |
| 6. |  |  |
| 6. |  |  |
| 7. |  |  |
| 8. |  |  |

|  |  |
| --- | --- |
|  | |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

## Classification

After we are able to visualize clusters and analyze their properties, we can start focusing on subsequent classification. Firstly we briefly analyze the work done in terms of cluster classification. Then, we discuss the problems and the choice of the classifier model. Another topic we examine is

Classification of a cluster is a task where we are given a cluster and a set of possible classes. Based on the cluster, we should predict to which class the cluster belongs. Ideally, we also want to estimate how sure we are about our prediction and possibly return the result "unclassified" if we are not confident about the prediction.

So far, there have been a few attempts to classify clusters – one example being the work Detecting elementary particles with Timepix3 detector [4]. In this work, the clusters were divided into classes based primarily on their shape. The categories used in the thesis were the following:

* Dot

[add images]

* Long gamma
* Heavy track
* Heavy blob
* Straight track
* Curly track

[add also example of work of Mr.Petr Manek]

Even though this classification provides valuable information about the shape of a cluster, mapping these categories to the real particle examples is still non-trivial. For instance, one particle can have a dot shape when it traverses the detector perpendicularly. In contrast, if the same particle enters the detector's field in a direction parallel with the orientation of sensors, it could leave a straight or even a curly track.

The categories we decided to use for classification were based on the training data we were provided by Mr. [insert title] Declan Garvey, working at the Institute of Experimental and Applied Physics of the Czech technical university. The classes of the particles in the data were the following:

* Electron
* Muon
* Pion
* Proton
* He
* Fe
* Fragmentation
* Lead

[add images for each class]

Because it seemed to be very difficult to manually set the criteria for each class, we decided to use a machine-learning-based approach. We chose neural networks as these became very popular when it comes to solving complex problems, and in many tasks, they achieve state-of-art performance. When fed with the data, the neural network model can learn from the data features until it reaches the maximum accuracy. There are many kinds of neural networks, but we narrowed the choice down to the two primary candidates – the convolutional neural network (CNN) and the multi-layered perceptron (MLP). The first candidate - the CNN - is widely used for the 2D image analysis, but it has a couple of drawbacks for our task. Firstly, the cluster would have to be represented as a 2D image containing both the pixels with nonzero energy and also the ones carrying no energy. This means using CNN could be a little less efficient because CNN processes all of the given pixels (even though the zero energy pixels provide no information about the cluster). Considering the fact that we are able to calculate features and work with those instead of the whole set of pixels lead us to conclusion that we could use the feature-based MLP model. [WIP]

*Training Data Generation*

Even though we had the data to train our model, the data were separated into files by particle type and angle of crossing the detector. This format can be great for viewing and browsing, but it is preferable to have one data source for training purposes.

# Development documentation

*Technology*

The framework I decided to use for this project is .NET Framework and Windows Forms. The reason for that is simply because my primary programming language is C#, and I consider Windows Forms to be reasonable for creating Graphical User Interface. However, WinForms does not support 3D graphical plotting by default. So, for this purpose, I will use an external data visualization library called Chart Director [11].

[should say how can the applications be extended, improved, how to create your own classifier]

## Important objects

*Cluster*

As we will be trying to work with clusters, we should create a Cluster object for each of these. This Cluster object will have the following properties:

* The First time of arrival
* Pixel count
* Byte of start in .px file
* Line of start in .px file
* Collection of its pixels, represented as an array of PixelPoint objects

Each PixelPoint should then contain its x and y coordinate, time over threshold and time of arrival.

## Viewer

*Browsing Cluster Collection*

Browsing of the collection of clusters is done via the buttons. Without the chance of loading the data into operating memory, we will need to search for a specific cluster sequentially each time we click previous. There is an option of creating a mapping table (index of a cluster, byte offset of a cluster) and then use .Seek() method (which might be faster, but will consume a significant amount of memory), but even if user used the viewer on the large input files and clicked 'previous' the sequential search still responds very quickly (.cl file is usually significantly smaller than .px file). Browsing of the .cl file is implemented via ClusterInfoCollection:IEnumerable.

*Skeletonization*

Skeletonization is the process of finding a skeleton of a binary image. For this purpose, I will use Zhan-Shuen's algorithm for thinning binary digital patterns [6] Because the image of the cluster we have is not binary, we assign a pixel with nonzero energy (Time over threshold) a value of 1 and a pixel with zero energy a value of 0. As the thinning process returns non-trivial data for bigger clusters, we will optimize the algorithm for those (~70-1000px). The key is to use well-fitting data structure. Throughout the algorithm we perform two operations most often. The first one is .Contains(), which means 'does the specific pixel have a value of 1?' and the second one is .Delete() with is setting the value of a specific pixel to 0. An ideal option for these operations seems to be an array of 256x256 pixels which can effectively do both these operations in constant time. The problem is that this approach would lead to huge performance issues as most of the clusters are smaller than 100 pixels and the initialization of the array with 65 536 items would take much time. Using the list collection List<T> from the standard library, we get rid of this problem. On the other hand .Contains() and .Delete() take linear time to execute. That is why we chose to use HashSet<T> which has similar memory usage as List<T> but both .Contains() and .Delete() are done in a constant time. In case we would need another performance boost, the whole algorithm can be almost trivially parallelized.

// histogram We need to iterate over the collection, and for that, we reuse ClusterInfoCollection.

//3d view For this purpose, we will use a Z-Calculator object to calculate the z-coordinate when provided with all of the necessary parameters.

## Filter

An abstract class ClusterFilter will represent a default filter for a given input. This abstract filter should provide a general 'Iteration method' – the way each filter traverses the data. This can be implemented by using a feature of C# called enumerator methods. The key words' yield return' could make the iteration method simple to implement. Each specific filter will then inherit 'iteration method' from ClusterFilter and add its own 'Predicate method' – a function that returns a Boolean value depending on whether a given cluster matches the criteria given by a filter.

*Complexity*

Because input files can be huge, it is necessary to make the processing algorithm efficient. Let  denote the size of the .px file and the size of the .cl file. The iteration process will have linear time complexity with respect to . 'Predicate method' time complexity depends on the specific filter. Filters that were implemented are based on pixel count, total cluster energy and convexity. Complexities: Pixel count is constant, the total energy is linear, and convexity is , where  is the pixel count of the -th cluster. Thus we obtain the upper bound for the time complexity of the filtering process Note: Time complexity can change after adding of new filters.

## Description Generator

## Classifier

# Experiment

## Baseline single-layered Model

## K-Fold cross validation

## Multi-layered classifier

**References:**

1. TimePix3 detector: <https://kt.cern/technologies/timepix3>
2. 3D reconstruction: <https://link.springer.com/article/10.1140/epjc/s10052-017-4993-4>
3. Clusterer and Mr.Meduna’s Work: <http://medunalukas.cz/utef/>
4. Mr.Meduna’s Bachelor Thesis(MM data format): <https://dspace.cuni.cz/handle/20.500.11956/106938?locale-attribute=cs>
5. 3D plotting library: <https://www.advsofteng.com/index.html>
6. Skeletonization: <http://agcggs680.pbworks.com/f/Zhan-Suen_algorithm.pdf>
7. Skeletonization <https://core.ac.uk/download/pdf/82669939.pdf>
8. Cluster Processor project repository
9. Skeletonization <https://www.researchgate.net/publication/308822048_A_new_thinning_algorithm_for_binary_images>
10. Accord.net: <http://accord-framework.net/>
11. Chart director library: <https://www.advsofteng.com/>
12. Skeletonization image <https://www.researchgate.net/publication/220273912_K3M_A_universal_algorithm_for_image_skeletonization_and_a_review_of_thinning_techniques>

**Epilogue / Conclusion**

[Sample: Bibliography is prepared in accordance with applicable standards. A recommended citation standard for the master thesis is ISO 690. Names of journals can be abbreviated, but only in a codified form. All used sources must be properly cited.]

**Bibliography**

[Sample: Tables in the master thesis, if any.]

**List of Tables**[Sample: Abbreviations used in the master thesis, if any, including their explanation.]

**List of Abbreviations**[Sample: Attachments to the master thesis, if any (various additions such as programme extracts, diagrams, etc.). Each attachment must be referred to at least once from one's own text of the thesis. Attachments are numbered.]

**Attachments**

[Sample: Hard back cover of the master thesis **- not a part of the electronic version**.]