2020

Guide for CIA

Proceedings

David Pérez González

22.12.2020

Contents

1. Aim of Proceeding

This document contains the guide for explaining how to use the Crystal Identification Algorithm (CIA) in Python. The document itself conforms to its own specifications. This guide is done to avoid possible issues with the use of this feature and to take care of the constraints that one needs to take into account.

2. Introduction

The following guide is based on the beta version of the CIA and on the version 2.7 of Python.

The goal of this algorithm is to increase the resolution of PET-detectors by identifying the different detected crystals and obtaining DOI information. To do that, four flood maps are generated from four different center-of-gravity (COG) algorithms. Each of them is splitted into smaller regions of interest (ROIs) where the peaks are concentrated. Then, a peak-finder routine is run through each ROI and the center-peak is labelled. With this label, the rest of the peaks from the ROI are as well identified going towards the edges. A double-check is applied later with all the ROIs together from the same flood map. Medians for the rows and columns are calculated and based on them, the peaks are confirmed as valid.

With all the labels from the four COGs we expect to have the complete spectrum of crystals identified. A LUT is created based on this information. All the clusters are then assigned to the crystals using the LUT. In principle, a homogeneous distribution of the clusters is expected, but some minor values from one of the COG algorithms are found both in this distribution and its corresponding flood map.

3. Obtaining Data

To obtain the positions of the clusters from the four different COGs, the hitAnalysis program (from mecforsatumanalysis) has to be run again with the corresponding "measurement.txt" file and in the folder where the data was recorded. In our case:

- nas/pet-scratch/Measurements/Hypmed/2019-09-20_-_15-25-35_-_Hypmed_Coinc/ 2019-09-20_-_15-25-43_-_first_tests/ 2019-10-18_-_15-52-50_-_3layersBaSO_08mm_2m/ ramdisks_2019-10-18_-_16-39-34/

The command itself is:

- hitAnalysis measurement.txt –workPackageBasedAnalysis –save-cluster-binary –save-cluster-ascii –no-plots –no-save

The possible filters or variable parameters are defined in the "settings_measurement.ini" file, located in the same folder. The filters that are activated are for minimum and maximum photon count and a filter of energy in some of the plots. A copy of the used file is presented in the Section Settings Measurement.ini.

In order to work in Python with the generated flood maps, we need to extract the data, namely the four different calculated positions ("x" and "y" coordinates) and a flag indicating whether they are valid or not, from the output file ".DebugSingles". In this case, from the measurement that was carried out on October 18 (2019) by Federica Demattè with a 3-layered crystal, an eight-millimeter light guide and two-meter cable. Therefore, we need to convert the files in a format accepted by Python, such as ".hdf5".

For that, a routine developed (from MonoCal) for this purpose was adapted. However, the routine cannot handle files that are too large like the one it was obtained with more than 100 million clusters. To overtake this problem, the file is split into 100 files of one-million clusters each.

This routine can also apply different filters to the data, but they were all deactivated. The file to do so is called "CreateData.ini" where one indicated the directory with all the files, the two output files (one for algorithm and the other one for testing) and the information contain on the input files. From the program "BasicCluster.py" one can define what can be extracted and how and with "CreateData_v3.py" the files are read and the data is selected, extracted and saved. These three files were modified to create the new routine based on MonoCal.

The commands for this procedure are the following (one needs to be in the directory where "CreateData_v3.py" file is):

- python
- from CreateData_v3 import CreateData
- CD = CreateData()
- CD._config_parser("CreateData.ini")
- CD.iterate()

After that, we have the two output files (".hdf5") with all the information we need to work in Python.

4. Loading Data

Now, we keep working on the MonoCal frame to load those files. However, some of the functions that are needed have been modified to adapt them to the current data. The main routine within this framework is called "Hypmed" and we need to import all the classes from it. The data will be loaded using a specific function and then, the clusters will be filtered by the validation flag and split into regions of interest (ROI). For every ROI a flood map is obtained and then saved with the library "pickle" which allows us to write on a file the python object.

These procedure has the following commands:

- python

- from Hypmed import *
- Hyp = Hypmed_Clustering()
- Hyp.sket.loadDataHyp("reference.hdf5", "test.hdf5")
- Hyp.valid_events()
- Hyp.region_ana()
- Hyp.Floodmap()
- with open('hist0_i.pickle', 'wb') as handle: pickle.dump(Hyp.hist0_all[i], handle, protocol=pickle.HIGHEST_PROTOCOL)

The histograms saved are lists of arrays where the first array is the number of entries in each bin defined by the second array ("x") and the third array ("y").

5. Analising Data

5.1. Peak Finder

Once we have the data saved on pickle files, we can now use a routine to find the peaks on the different histograms. It is based on the ROOT routine "SearchHighRes" and for every COG algorithm the parameters are changed (sigma, threshold, background and iterations of the convolution). The chosen values are shown in the following:

- sigma = [3.3, 2.5, 1.5, 1.5] -> [000,010,100,111]
- threshold = [2, 6, 7, 4] -> [000,010,100,111]
- rmBackground = True -> forall
- convIter = 200 -> forall with more tends to reagroup peaks and with less some are missing
- markov = False -> forall smoothing function to remark peaks, but works for us, peaks with different heights and sigmas
- mIter = 3 -> different number of iterations have been tried

The output is an array of the "x" and "y" coordinates from the found peaks. Those coordinates are then ordered according to the "y" axis.

5.2. First Part of CIA

The first 20 peaks (clusters) are taken to be ordered by "x" axis. The closest to the left top corner is chosen as reference for the row. The next one must be within a range (a different one for every COG position algorithm): "y" position of previous peak +− 0.1-0.2 mm. If not, the peak is saved for the next iteration. A scheme of this iteration is shown in Figure 1. This process is repeated 20 times because no more than 20 rows per ROI are expected. In case a peak or more could not be classified, they are saved for the second stage of the CIA in a dictionary.

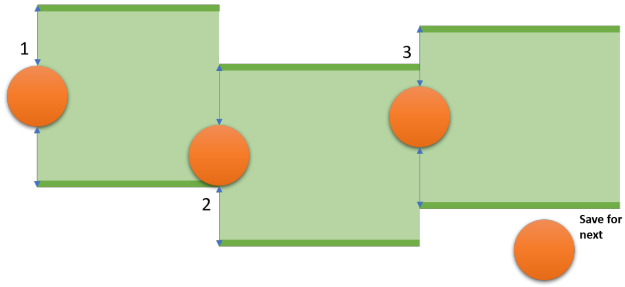

Figure 1: Scheme of iteration for obtaining rows.

For every histogram a dictionary of rows is created. The label reference is the closest peak to the center of the ROI. This peak is assigned as it is the best defined peak and we know exactly to which label belongs to. Then, the rest of the peaks belonging to the same row are assigned towards the edges. An example of assigning rows is shown in Figure 2.

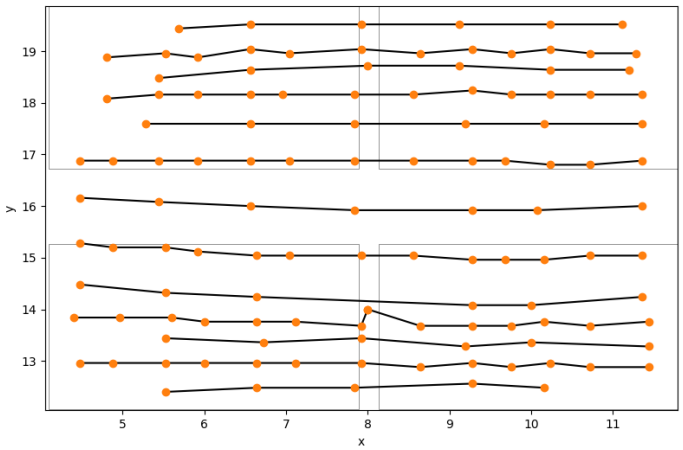

Figure 2: Classification of rows from one ROI of 111-COG.

For example, if the peak in the center has the label 835, the label for the peak next to it (increasing the value of "x") in the same row would be 836. On the other direction,

it would be 834. Then, the closest peak to this reference from the next row considering the "x" coordinate would be the reference label for the corresponding row. Here, we select how many rows and columns are accepted as valid pro ROI depending on the flood maps.

This information is saved in the general dictionary for the corresponding COG, where all the labels are variables of the dictionary. In case one peak is assigned to one label, this information is saved to the dictionary together with the layer to which belongs. In case there was already a peak from a different ROI, the new peak is appended. This case is known as ghosting, a peak which appears in two consecutive ROIs due to the fact that the crystal is between the regions.

To distinguish the peaks from the different layers, we can also use the reference as a guide. We know to which layer it belongs to and we know that if it is the third layer, the next one would be also belonging to the third layer and the reference for the next row would be a peak from the second layer. Next to the peak from the second layer, we have a peak from the first layer and so on. The general expected distribution of the peaks are shown in Figure 3.

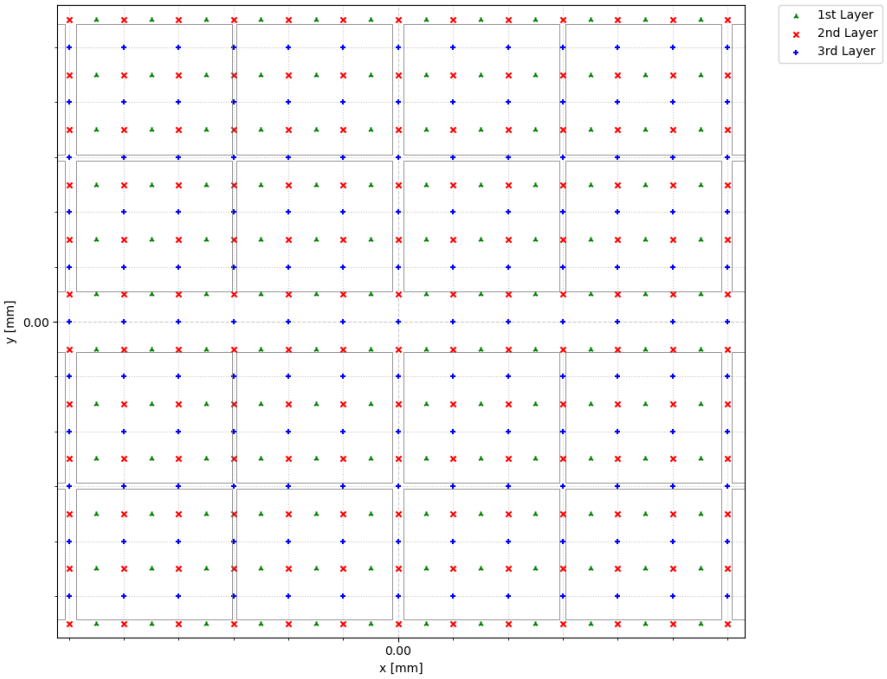

Figure 3: Expected distribution of peaks from different layers.

Once we have assigned all the peaks from the dictionary of rows, the next ROI is iterated. After all the ROIs from the corresponding COG are obtained, a file is created with the dictionary including all the labels. A flow chart representing this procedure is shown in Figure 4.

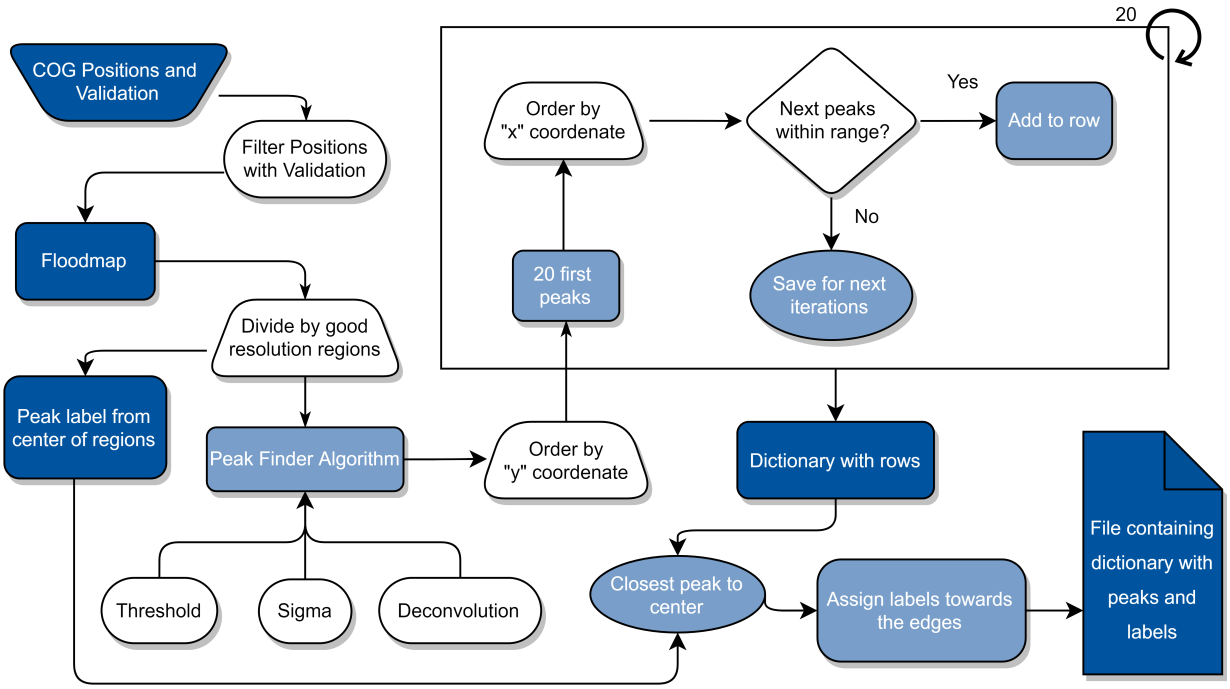

Figure 4: Flow chart of the first part of the CIA.

5.3. Second Part of CIA

After the first classification, some of the peaks could be not correctly classified (as we can see in Figure 2) or directly not classified. That is why, a second check is done. Firstly, all the classified peaks from all the ROIs are taken and the median of the column from those valid ones belonging to the same column is calculated. In case they are within a specific range: median of column +− 0.2-0.3 mm (depending on which COG we are), they are confirm as valid. When that is not the case, the peak is marked as invalid.

The same procedure is followed with the medians of the rows. Now, all those peaks that are not valid, are then assigned to the closest column and row. If the new label is empty, the peak is assigned to it. If the label is invalid, we double check again the corresponding old and new peaks. Individually, each is checked whether they are within the specified range and then marked as valid.

At last, those peaks that did not have a label are also analysed. Their closest row and column are calculated and the same check is done. Finally, the new dictionary with the corrected labels is saved again in a file.

Now, in Figure 5 we can observe again the same ROI as shown in Figure 2, but with the correct classification of all the peaks.

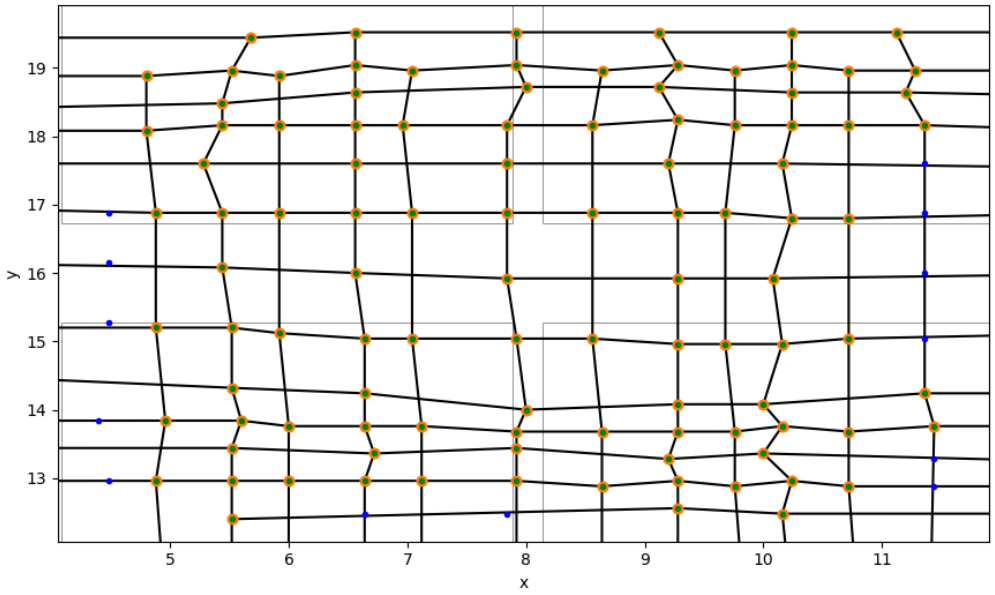

Figure 5: Final classification of peaks with columns and rows. The orange dots are valid peaks and the blue dots are labels with two peaks.

An example for a dictionary element follows:

dic_crystal[1715] =
{'center': 12: [[3.359644, 0.001895]], 13: [[4.560458, 0.00071]], (12 and 13 are ROI's id)
'id': 1715,
'layer': 3,
'row': 35,
'valid': True}

A simplified flow chart of this part is shown Figure 5.

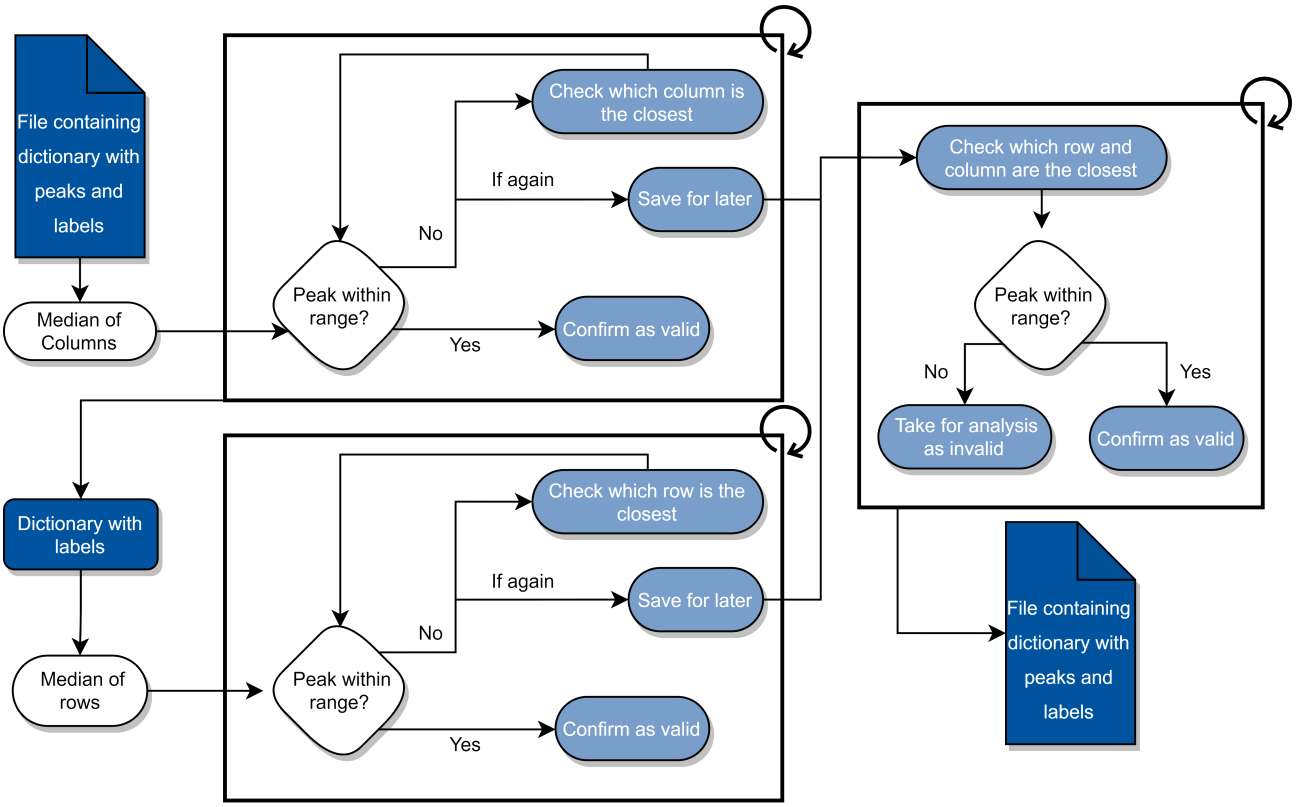

Figure 6: Flow chart of the second part of the CIA.

6. Cluster Identification

Once the four COGs algorithms are passed to the CIA and four different dictionaries of labels are created, a cluster identification can be carried out.

To do so faster, a look up table (LUT) is created for all of them. A grid with a resolution of 0.1 mm is created for each COG algorithm. Each coordinate is then analysed to look for the closest label from the dictionary. The two closest peaks are saved in the LUT together with their relatives distances. If the distance is larger than 1 mm, the coordinate is not assigned to improve the classification later.

Example of lut element:

lud111[0,0] =
{'2CLOP': 'center': 12: [[-0.160166, -0.957595]],
'id': 1760,
'layer': 2,
'row': 36,
'valid': True,
'CLOP': 'center': 12: [[-0.160166, 0.001895]],
'id': 1712,
'layer': 3,
'row': 35,
'valid': True,
'QF': 0.026496736254681918}

Now, every single cluster can be round to 0.1 mm of resolution. These coordinates are looked in the four different LUTs from the COGs. If they have a label assigned and it is valid, the label is saved for comparison. When the valid labels agreed, the label is assigned to the cluster. When not, a coincidence between two or more of them has the priority to assign the corresponding label to the cluster. If none of them agreed, the quality factor (QF) extracted from the relative distances of the closest peaks has the priority. The quality factor is defined as:

$$QF = \frac{lowest\ distance}{lowest\ distance\ +\ second\ lowest} \tag{1}$$

The lower the QF, the better the peak is, because it means the relative distance to the first peak is smaller than the relative distance to the second peak and, that means, that in this COG this cluster is clearly identified to one peak. When the distances are similar, it means that the cluster is not clearly identified with only one peak. Thus, it is a worse identification.

A test of the cluster classification has been carried with 75 million clusters. The number of clusters assigned to each label is represented in Figure 7. Some of the crystals have more clusters assigned as others. On the edges it is normal to have more clusters assigned to the crystals because as we can observe some of the first and last rows are missing. They are not identified because they are collapsed in the flood maps and

therefore, they are not distinguishable. The other differences encountered could be due to the different filters applied. A deeper look into the data is needed to check whether there are problems in the classification.

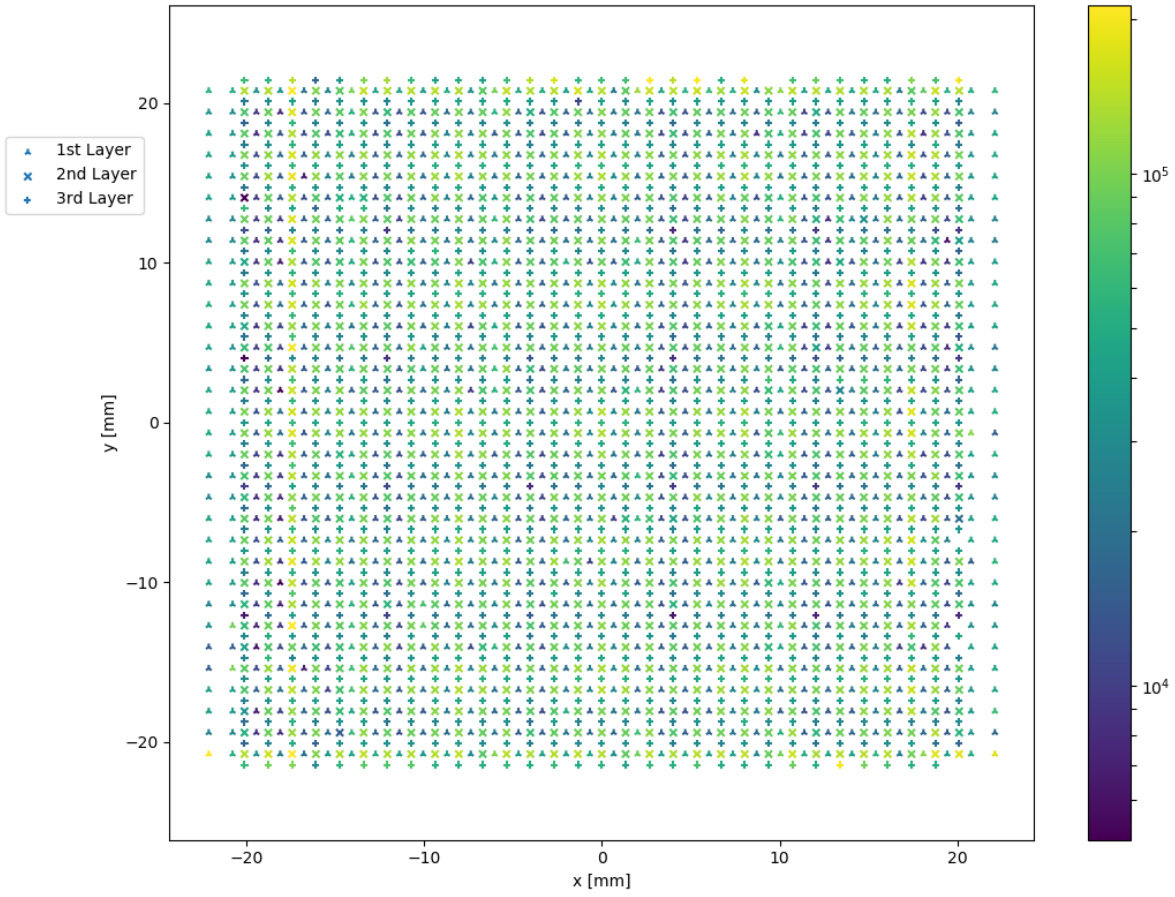

Figure 7: Number of clusters in each label in logarithmic scale. Those with 0 clusters are not represented (first and last rows).

A. Settings Measurement.ini

```
#[frameCounter]
#bitWidth=8 #we use the 32 bit mec framecounter and thus do not need to correct the 8bit pdpc framecoutner anymore

[dieConfiguration]
inhibitFraction=0.1

[clock]
frequency_MHz=195

[recPars]
randomizeSinglesInCrystalBin=false

#a lot of the following parameters can be optimized by using a special result extraction method that produces a debug print for the memory used by each workpackage (commandline options "--workPackageBasedAnalysis --analyseWorkPackageSize")
#this is the number of frames that are put into one workpackage (this should be 100 if raw data processed by the daps is used as the daps puts it also in 100 frame junks)
workpackageSize=50
#the number of reading threads is equal to the number of inout files (for MEC scanner it is usually 10)
#the following memory pool sizes are for each reading thread
#this is the size (byte) of one memory pool storing msg objects (without the message content)
readingThread_memory_msgs_poolSize=4000000
#this is the maximum number of memory pools for messages a single reading thread is allowed to allocate (if reading is faster than processing it will allocate this amount)
readingThread_memory_msgs_maxNumberPools=5
#the following two values are again for reading threads but this time for the memory used to store message content
readingThread_memory_msgMemory_poolSize=10000000
readingThread_memory_msgMemory_maxNumberPools=4
#this is the number of processing threads that are started
#processing threads will work on workpackages filled by the reading threads with the raw messages and will go through the complete data processing
#the number of threads can be chosen from 1 up to any number (it makes sense to keep it a bit smaller than the number of available cores [reading threads and result extraction thread(s)])
numberOfProcessingThreads=6
#the following values define the memory pool size and maximum number of memory pools per processing thread for hit, cluster and coincidence objects
processingThread_memory_hits_poolSize=8000000
processingThread_memory_hits_maxNumberPools=5
processingThread_memory_clusters_poolSize=32000000
processingThread_memory_clusters_maxNumberPools=5
```

```
34 processingThread_memory_coincidences_poolSize=1000000
35 processingThread_memory_coincidences_maxNumberPools=5
36 #result extraction (e.g. listmode files and plotting) is coordinated by a
       single thread that work on completely processed workpackages.
37 #this thread will schedule plotting jobs for seperate threads to multi
       thread result extraction, this is not configurable atm




41 #number of hits to be processed if the --max-hits option is set (is not
       100% accurate)
42 NumberHitsToProcess=5e9
43 #maximum number of ml poisson iterations
44 MLPoissonMaxIterations=5
45 #cluster window / ps
46 clusterWindow=40000
47 #coincidence window / ps
48 coincidenceWindow=2000
49 #method for the anger algorithm. possible values: allChannels,
       mainPixAndNeighborPixels, mainPixAndNeighborDies/onlyNeighbors,
       circleAroundMainHit, sameQuadrant, mainPixAndDirectNeighborPixels,
       mainPixAndAutoNeighborPixels
50 angerMethod=mainPixAndNeighborPixels
51 #if anger mehtod is circleAroundMainHit then this defines the radius
       around the main hit:
52 angerMethod\angerRadius=10.
53 #use the following method to identify the crystal: angerMap (default, if
       ommited), oneToOneCoupling
54 crystalIdentificationMethod=angerMap
55 #use a simple logarithmic saturation correction
56 photonCountCorrection=saturation_log
57 #apply the pixel gain if it has been calibrated
58 usePixelGain=true
59 #photon counts used for energy calculation. possible values: allChannels,
       mainAndNeighborPixels/onlyNeighbors, mainAndAutoNeighborDiesSum,
       mainAndNeighborDies, mainAndDirectNeighborDies, mainDie, mainPix,
       mlPhotonSum
60 energyCalculationMethod=mainAndNeighborDies

62 #cluster filter definitions: you can define cluster filters at different
       stages of the processing chain
63 #  -there are cluster filters used only for plots which do not influence
       the processing at all
64 #   these filters are bound to a certain plot in the main.cc
65 #   /clusterPlots - the standard filter used for clusterPlots using a
       filter
66 #   /clusterPlots/[subIndex] - atm only index 0 is used for a filter
       without an anergy window
67 #  -/beforeCrystalIdentification - can be used for a fast photon cut (no
       neighbor information!!!! or energy is available !!!!!!)
68 #  -/clusterFile - is only used directly before piping clusters into the
       cluster binary file per tile, the beforeCrystalIdentification  has
```

```
    already been applied
69 #  -/beforeCoincSearch - is applied before searching for coincidences,
    this may influence the random rate, if you filter before the coinc
    search as you may have a triple which looses one cluster
70 #  -/coincSearch - is applied to coincident clusters, if no filter is
    applied before the coinc search one can reject all multi events
71 #
72 #possible filter settings (the /beforeCrystalIdentification does only
    support the photon cuts !!!!):
73 #  name: string with name
74 #  minPhotons, maxPhotons: integer values
75 #  minEnergy, maxEnergy: floating point values
76 #  neighborFilter: filter on the presence of the neighboring pixels of the
     main pixel; string value: none, directNeighbors, diagonalNeighbors,
    allNeighbors
77 #  minMainPixRatio: the minimal rewuired ratio of the photon value in the
    main pixel to the total number of photons in a cluster; floating point
    value [0., 1.]
78 #  maxAngerSigma: maximal distance of a cluster to the nearest crystal
    anger position in terms of sigma; floating point value e.g. 3.

80 #clusterPlots: these are used for cluster plots and do not influence the
    processing, the mapping to plots is hardcoded in the main.cc!!
81 clusterFilter/clusterPlots/name="standard"
82 clusterFilter/clusterPlots/minPhotons=500
83 clusterFilter/clusterPlots/maxPhotons=5000
84 #clusterFilter/clusterPlots/minEnergy=411
85 #clusterFilter/clusterPlots/maxEnergy=561
86 clusterFilter/clusterPlots/neighborFilter=allNeighbors
87 clusterFilter/clusterPlots/0/name="noEnergy"
88 clusterFilter/clusterPlots/0/minPhotons=500
89 clusterFilter/clusterPlots/0/maxPhotons=5000
90 clusterFilter/clusterPlots/0/neighborFilter=allNeighbors
91 clusterFilter/clusterPlots/1/name="noEnergy_direct"
92 clusterFilter/clusterPlots/1/minPhotons=500
93 clusterFilter/clusterPlots/1/maxPhotons=5000
94 clusterFilter/clusterPlots/1/neighborFilter=directNeighbors
95 clusterFilter/clusterPlots/2/name="noEnergy_diagonal"
96 clusterFilter/clusterPlots/2/minPhotons=500
97 clusterFilter/clusterPlots/2/maxPhotons=5000
98 clusterFilter/clusterPlots/2/neighborFilter=diagonalNeighbors
99 #beforeCrystalIdentification
100 clusterFilter/beforeCrystalIdentification/name="
    beforeCrystalIdentification"
101 #clusterFilter/beforeCrystalIdentification/minPhotons=1000
102 #clusterFile
103 clusterFilter/clusterFile/name="clusterFile"
104 clusterFilter/clusterFile/neighborFilter=allNeighbors
105 clusterFilter/clusterFile/minPhotons=500
106 clusterFilter/clusterFile/maxPhotons=5000
107 #clusterFilter/clusterFile/minEnergy=
108 #clusterFilter/clusterFile/maxEnergy=
```

```
109 #beforeCoincSearch: this filte ris applied BEFORE the clusters are
        processed by the coincidence finder
110 clusterFilter/beforeCoincSearch/name="beforeCoincSearch"
111 clusterFilter/beforeCoincSearch/minPhotons=500
112 clusterFilter/beforeCoincSearch/maxPhotons=5000
113 #clusterFilter/beforeCoincSearch/minEnergy=411
114 #clusterFilter/beforeCoincSearch/maxEnergy=561
115 clusterFilter/beforeCoincSearch/neighborFilter=allNeighbors
116 #filter for coincidences: minTiles, maxTiles, minLORLength (/ mm),
        minTileDistInPhi AND/OR minSPUDistInPhi (USE ONLY FOR MEC SCANNER WITH
        10 SPUS!!!!!) (specifies the minimum number of tiles/SPUs in BETWEEN
        the two coincident tiles/SPUs)
117 coincidenceFilter/minTiles=2
118 coincidenceFilter/maxTiles=2
119 #coincidenceFilter/minLORLength=500
120 #coincSearch: this filter is applied after the coincidence search to every
         cluster of a valid coincidence.
121 #            You only need to implement stricter filters than the filter
        vefore search, beacause all clusters must have already passed the
        prviously applied cluster filter
122 clusterFilter/coincSearch/name="afterCoincSearch"

124 [sinogramPars]
125 # Half the crystal width thanks to intercept theorem:
126 # The distance between the two LORs L_0,300 and L_0,301 is 0.5 mm
127 # at half of the length of the LORs for a crystal width of 1 mm
128 displacementResolution=2
129 # Inner diameter
130 displacementSize=217.6
131 # sin(pitch/radius) * 180/pi
132 # radius instead diameter to prevent checker board pattern
133 angularResolution=2.1
134 NRings=8
135 # These are the upper edges of the axial crystal bins
136 ringsLUT=-12, -8, -4, 0, 4, 8, 12, 16
```