C. Characterization of the clusters

This appendix documents how the code for doing the preprocessing, clustering and characterization of the Timepix3 data can be executed by walking through the procedure on a small test data set. The code is written in Python with ROOT, therefore both Python and ROOT needs to be installed. ROOT can be downloaded from `https://root.cern.ch/`. The code was run with Python verison 2.7.14 and root version 6.10.

The input to the analysis is the raw data from the Timepix3 detector in ascii files, and the analysis is done in three steps. The first step is the energy cut and the clustering, the second step is to write the clusters to ROOT files, and the third step is to do the characterization of the clusters.

The output of the whole analysis is a file containing the following information for each cluster: cluster size, number of prongs, the χ^2 square value of the linear fit, deposited energy in the whole cluster, and deposited energy in the center of the cluster.

The code for the data analysis is found in the git repository `finalTimepix`, where a small data set is included for test purpose. To execute the code clone the repository and checkout tag `v1.1`, navigate to the `dataAnalsyis` folder, and check that all files are there:

```
git clone https://github.com/helgaholmestad/finalTimepix.git
git checkout v1.1
cd finalTimepix/dataAnalysis
ls
checksSimularity.py       compareReferenceMain.py
hough1D.pyc               loopClusters.py
loopReversed.py           preprocess.py
runReversed.sh            clusterinNotParalell.py
hough1D.py                illustrateCharaterization.py
loopHough.py              makeStatistics.py
runHough.sh               testData
```

The folder `testData` contains a small part of the full data set from the Timepix3 detector. The data files from the Timepix3 detector are ascii files named `data_<i>.dat`, where `<i>` is the file number, and a single file corresponds to one spill of antiprotons. A data file has in total 8 columns, and the relevant fields are explained in table C.1.

Step one is to do the halo cut and the clustering. This can be done with the Python script `preprocess.py` and the path of the folder containing the data as the input parameter.

```
python preprocess.py testData
```

This script makes a new file named `clustering<i>.dat` for each input data file and

Column	Description
1	Pixel number in x
2	Pixel number in y
5	Deposited energy in keV
6	Time of arrival in ns

Table C.1.: Description of the relevant fields in the Timepix3 raw data files.

places it in the same sub folder as the input data file[1]. In the clustered data there is one one row for each pixel, only the relevant fields are kept and the data is sorted by the clusters. Each cluster starts with the line `new cluster <T>`. The variable T gives the time of arrival in ns of the first peak in the time of arrival distribution of the relevant spill, therefore this number is the same for all clusters from the same spill. The definition and explanation of the two different peaks in the time of arrival distribution is found in section 4.4.1. Below is an excerpt from a clustered data file:

```
new cluster 146585.0
0 226  8.006  148387.6047
1 224  26.936  148390.1047
1 223  22.975  148392.7246
0 224  19.661  148389.1008
2 223  28.243  148393.1605
0 225  53.921  148390.6601
2 222  41.897  148393.2563
new cluster 146585.0
55 30  8.618  148418.4401
55 31  47.854  148421.9326
54 31  7.68  148419.93
new cluster 146585.0
52 90  35.473  147191.5262
52 88  59.896  147192.2093
52 91  40.387  147194.8193
51 93  23.513  147198.4682
52 89  53.334  147191.9703
```

Step two is to put the clustered data into ROOT files using the Python script `loopClusters.py`, also with the name of the folder containing the data as the input parameter. Here also the time cut described in section 5.2 is performed, as only the clusters passing the time cut is written to file.

```
python loopClusters.py testData
```

Storing the data in ROOT files is beneficial because they are faster to read from than ascii, and one can inspect them visually by using a TBrowser. In the ROOT files there is one histogram per cluster, and one example of a cluster is seen in figure C.1.

[1]The program takes quite some time, and the script paralellClustering.py shows an example how how the process can be parallelized using the Python library parallel python. However, this script only runs with Python2 as this specific library is not available in Python 3.

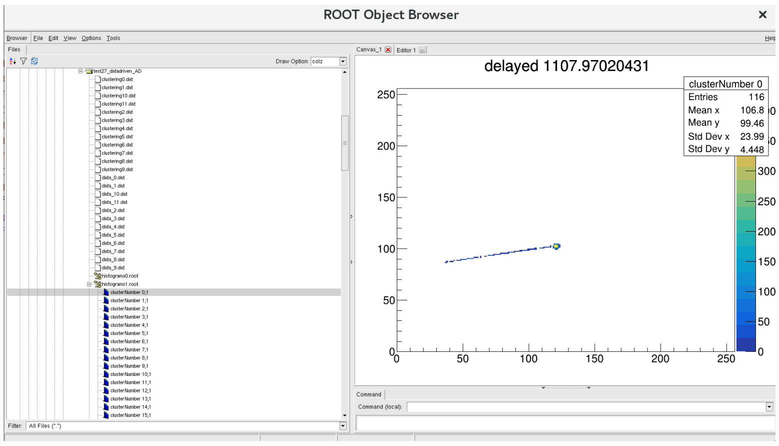


Figure C.1.: Example of how the clusters can be inspected using a TBrowser.

Step three is to do the characterization as described in section 5.4. The information about the clusters is written to a text file placed in the folder `datafiles`. This folder must be made before the program is executed, and the program is excecuted as:[2]

```
mkdir datafiles
python loopHough.py testData
```

The outputfile is found in the folder `datafiles` and has the name `meta<dataFolder>.txt`, in this example the name of the file will then be `metatestData.txt`. This means there is one output file per folder of input data. Since one folder of input data often contains all data from one experimental setup, this is a convenient way naming the output data. Below is an excerpt from the file `metatestData.txt`:

```
newCluster
energy 456.977
pixels 88.0
prong 2
prongLenght 1784.74788135  1311.95464861
clusterCharge 2097.256
error 1.08688638309
trough
newCluster
energy 1318.362
pixels 26.0
prong 0
prongLenght
clusterCharge 1871.285
```

[2]There will be some warnings from the Python library polyfit, claiming that the data is not good for a linear fit. These warning is not an actual problem since they might appear when the pixels making up the cluster is not appropriate for making a linear fit. For the majority of the clusters the pixels don't make up something close to a straight line, the linear fit is made to tag the cluststers where this indeed is the case since those clusters are unlikely to be annihilation clusters. This is described in section 5.4.

```
error 0.737368342684
notTrough
newFile
newCluster
```

The line `newCluster` marks that there is a new cluster, and then follows all the information about that clusters. The line `newFile` indicates that also a new file is processed, and in the current data structure this corresponds to a new spill from AD. The explanation for all the cluster variables is found in table C.2.

Variable name	Description
energy	Energy deposited in the center of the cluster [keV]
pixels	Number of pixels in the cluster
prong	Number of prongs in the cluster
clusterCharge	Energy deposited in the whole cluster
error	The χ^2 value of the linear fit of the cluster
trough/notTrough	Indicates if the cluster got through the 70 pixel and 1 prong cut

Table C.2.: Explanation of the output variables in the metaTestdata.txt file.

D. Simulation of the annihilation events

This appendix walks trough how an ensemble of 100 simulated clusters is made. The simulation software FLUKA is used to generate the raw energy depositions that will be post processed, and is therefore a prerequisite. FLUKA can be downloaded from `http://www.fluka.org/fluka.php`, and installed according to the instructions on the web page. For this example and the in the work presented in this thesis, the FLUKA version fluka-2011.2x-1.x86_64 installed via RPM was used. The code for post processing is written in Python with ROOT, therefore both Python and ROOT needs to be installed. ROOT can be downloaded from `https://root.cern.ch/`. The code was run with Python version 2.7.14 and ROOT version 6.10.

D.1. Creating the simulated clusters

The first step is to use FLUKA to simulate the raw energy depositions. Start by cloning the `finalTimepix` repository if it is not done, checkout the tag `v1.1` and navigate to the `runningFLUKA` folder:

```
git clone https://github.com/helgaholmestad/finalTimepix.git
git checkout v1.1
cd finalTimepix/simu/runningFluka
ls
instructionsTruth.txt  supersimpelTimepixCenterCa.inp
instructions.txt       supersimpelTimepixCenterCalium.inp
makeTruth.py           supersimpelTimepixCenterGold.inp
mgdraw-pix.f           supersimpelTimepixCenter.inp
testPion.inp           supersimpelTimepixLayers.inp
timepixExample.inp
```

The file `timepixExample.inp` defines the detector, the impinging antiprotons and the voxels where the energy depositions is recorded. To run the FLUKA simulation do:

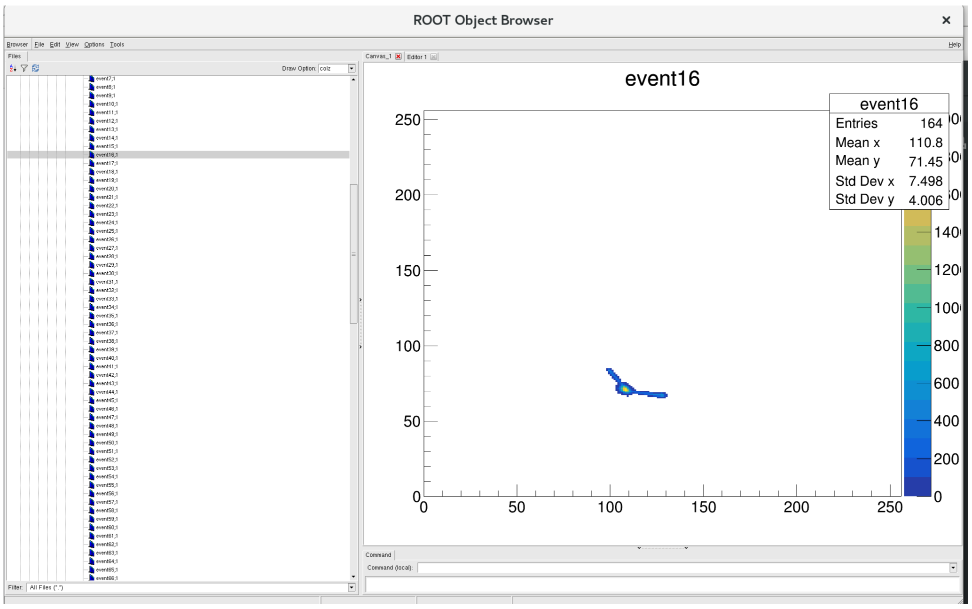
```
$FLUPRO/flutil/rfluka -N0 -M1 timepixExample.inp
```

This will create a file named timepixExample001_fort.22 This file contains information about the energy deposited in the voxels for each annihilation event, and is the file that will be post processed.

The first step of the post processing is to simulate the charge sharing by doing a Gaussian blur. The folder for the processed files is named `datafiles` and should be created before the post processing starts. Navigate one folder up to the `simu` folder, create the `datafiles` folder and run the Gaussian blur with the FLUKA output as the first input parameter. The second input parameter just makes an extension to the name of the output file and is used when running many files in parallel.

```
cd ..
pwd
/pathToRepo/finalTimepix/simu
mkdir datafiles
python GaussianBlurTCAD.py runningFLUKA/timepixExample001_fort.22 1
```

This program creates a ROOT file `histogramsTCADRaw1.root` in the folder `datafiles`. The ROOT file contains histograms of the simulated annihilation clusters. There is one histogram per cluster, one bin per pixel and the content of the bin is the deposited energy in that pixel. These histograms can be inspected by using ROOT's `TBrowser`, and one example is shown in figure D.1



event16	
Entries	164
Mean x	110.8
Mean y	71.45
Std Dev x	7.498
Std Dev y	4.006

Figure D.1.: Example of one simulated cluster

The next step is to take the volcano effect and the suppressed pixels into account as explained in section 6.1.2. For this example the data used for sampling the suppressed pixels is taken from the test data set. Therefore the clustering of the test data should be done before this step can be carried out. This is first step of the characterization as discussed in appendix C. The step is done by running the command `python preprocess.py testData` in the `dataAnalysis` folder, and if it is not done the program `reCluster.py` will be stuck in an infinite loop.

The code for simulating the volcano effect and the suppressed pixels is in the file `reCluster.py`. The input parameter is the extension to the file name used for the Gaussian blur. Since the suppressed pixels might divide one cluster into two or more, this code includes doing the same clustering as was done in appendix C for the real data.

```
python reCluster.py 1
```

The output of this code is the file `histograms1TCADFinal.root` in the `datafiles` folder. This file contains the histograms of the simulated clusters when the charge sharing, the volcano effect and the suppressed pixels has been taken into account.

To do the analysis to find the cluster variables the program `hough1D.py` is ran. The input parameter to this program is the ROOT file containing the clusters and a unique name to identify the output files.

```
pwd
pathToRepo/finalTimepix/simu
python hough1D.py datafiles/histograms1TCADFinal.root datafiles/meta
```

The `meta.txt` file has exactly the same format as the output file of the analysis of the real data as shown in appendix C. In addition a file named `metaprong.txt` is produced in the folder `datafiles`. This file contains information about the detected prongs in the cluster. The format is the following:

```
new cluster
center 198.311141025  172.796297525
pixel  198  178  56.37972298
pixel  198  179  118.889035307
pixel  198  180  188.773777375
pixel  198  181  215.906299437
  |     |    |     |
  |     |    |     |
pixel  208  206  5.21709037617
pixel  208  207  8.67553626524
pixelsInProng  103
pixel  197  166  27.1936264799
pixel  197  167  136.187449646
  |     |    |     |
  |     |    |     |
pixelsInProng  41
numberOfProngs 2
done
new cluster
```

The first line that starts with `center` gives the mass center. The positions is given as the pixel number such that the center of the pixel in the lower left corner is at position (1,1). Then follows information about the pixels making up the prong, first the position and then the deposited energy in keV in the pixel. Each prong ends with information about the number of pixels making up this prong, and each cluster ends with information about the number of prongs in the cluster. This output file will later be used to reconstruct the annihilation point by the vertex fitting method.

D.2. Extracting truth information

In order to access the truth information about the annihilation point the user routine MG-DRAW needs to be compiled, and then FLUKA should be ran with this user routine.

```
cd /pathToRepo/finalTimepix/simu/runningFluka
$FLUPRO/flutil/ldpmqmd -o exe mgdraw-pix.f
$FLUPRO/flutil/rfluka -e exe -N0 -M1 timepixExample.inp
```

The truth information is written to the `timepixExample001.log` file. A small Python script writes the truth information to a nicely formatted text file taking the log file and name of the output file as input parameters

```
python makeTruthExample.py timepixExample001.log truth.txt
```

The output file `truth.txt` has the format:

```
0 191.171365694  158.393974362
1 191.348774523  230.587351091
2 194.776192675  222.238588834
3 35.756368491  195.160570002
4 20.8085729225  189.538836243
5 245.867519133  33.5219841569
6 235.849257887  97.9953209373
```

The first column is the event number and then follows the x and y position of the annihilation point. The positions is again given as the pixel number such that the center of the pixel in the lower left corner is at position (1.0,1.0).

D.3. Reconstructing the annihilation point

A small example of how the do the vertex reconstruction on the simulated data is included. Navigate to the vertex fitting folder and run the program `vertexFittingExample.py` with the file containing the prong data and a chosen name for the output file as input parameters.

```
cd /pathToRepo/finalTimepix/simu/vertexFitting
python vertexFittingExample.py ../datafiles/metaprong.txt results.txt
```

The output file has the following format: For each cluster it contains information about the mass center, and then the reconstructed annihilation point for each possible combination of two prongs is given. The positions is as before the pixel number such that the center of the pixel in the lower left corner is at position (1.0,1.0). The fourth column is the angle in degrees between the two prongs that are used to reconstruct the relevant annihilation point.

```
new cluster
massCenter  22.0000646174  190.061284111
vertex 29.1588593286  187.624872797  85.0049825179
vertex 21.6411245815  190.748511311  59.0369651056
vertex 21.3641502937  190.863594854  71.2237611531
vertex 23.7976664964  189.852462486  26.0654535817
vertex 22.3587103983  189.777827138  54.0419476235
vertex 18.2857278167  191.06734971  76.2187786352
vertex 23.3548393875  189.462448717  31.0604710639
vertex 21.5659208194  190.850240052  40.2607262587
vertex 22.8927309522  189.055453191  4.89758131267
vertex 24.6953758855  190.643107447  44.8416924286
new cluster
```

```
massCenter  246.148465667  34.4935013328
vertex 246.917680894  34.4842538631  8.63975752901
vertex 246.145371006  33.9513295562  52.6518560015
vertex 246.558949037  35.2208249113  45.9879015275
new cluster
```