DoubleFit Manual
(translated version)
Version 1.0

Matthias Enno Janssen

26.01.2004

Contents

1 Introduction **2**

2 Structure and Operating Mode **3**
2.1 The Input Format . . . 3
2.1.1 ROOT: Tree h5000 and h5001 . . . 3
2.1.2 LCIO: TPCPulse . . . 4
2.2 The Modules . . . 4
2.2.1 ClusterFinder . . . 4
2.2.2 TrackFinder . . . 8
2.2.3 TrackFitter . . . 9

3 Usage **11**
3.1 The Configuration File . . . 11
3.1.1 Pad↔Channel Mapping . . . 13
3.2 The Display Function . . . 15
3.2.1 Screenshotmode . . . 15

Chapter 1

Introduction

Not yet translated.

Chapter 2

Structure and Operating Mode

Part not yet translated.

2.1 The Input Format

As already mentioned there is the possibility to choose between two input formats. With `InputFormat` it is chosen between ROOt and LCIO format [1].

2.1.1 ROOT: Tree h5000 and h5001

In the ROOT files produced with mktree or ntmake?? and h2root both trees h5000 and h5001 are included. Tree h5001 contains additionally the information of N (N¡=10) time slices before and after the pulse which is stored in h5000. The number N is calculated automatically in DoubleFit. In h5001 some variables are named differently.

The contents of both trees are described in the following, whereas the variable name used in tree h5000 is listed first. In the case that no second name is given, both names are the same; "-" means that the variable is only implemented in h5001.

trigger trigger number (Int_t)

runtime time in seconds, which has passed from start of the run until the recorded event

nchan,npulse number of pulses recorded (Int_t)

chan,pad channel number for each pulse; it is translated into the pad number using the right mapping (Int_t[npulse])

nslice,ntimeslice number of time slices of each pulse (Int_t[npulse])

time time of the pulse, given in time slices (int_t[ntpulse][10]), only read out until ntimeslice

charge number of FADC counts in the time slice (int_t[ntpulse][10]), only read out until ntimeslice

-,timeminus number of time slices before the pulse (int_t[ntpulse][10]), only read out until N

-,chargeminus FADC counts before the pulse (int_t[ntpulse][10]), only read out until N

-,timeplus number of time slices after the pulse (int_t[ntpulse][10]), only read out until N

-,chargeplus FADC counts after the pulse (int_t[ntpulse][10]), only read out until N

2.1.2 LCIO: TPCPulse

2.2 The Modules

The program is built up in a modular way, each module can work separately, i.e. it can be switched on and of separately in the config-file. Each module has its own output file, which is used as input file for the following module (see Fig. 2.1). They contain additionally all

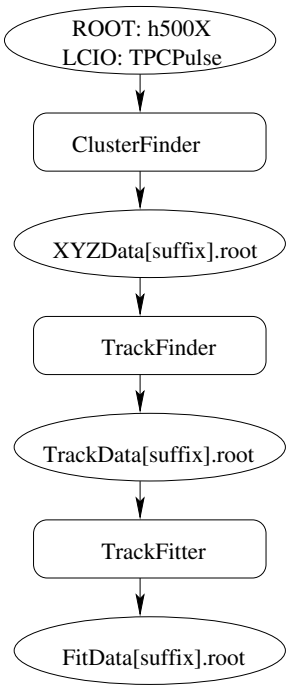


Figure 2.1: The modules and their outputs

information which was stored in the previous file, but only the events which have passed the requirements, e.g. a certain number of reconstructed hits per track (seesec:TrackFinder).

All data files produced by DoubleFit have one common suffix, which is defined in the config-file. It is useful to create different files and suffixes for different set of applied cuts. Then afterwards it can easisy be reconstructed which set of cuts produced a certain output file.

2.2.1 ClusterFinder

The module ClusterFinder reads data from the files which are indicated in the config-file (`InputDataFiles`). These must be given in the format `InputFormat`. In the case that the input format is ROOT, either the tree h5000 or the tree h5001 can be chosen according to the switch `CFTreeName`. The output format contains for each event:

NumberOfPoints number of recorded points

DataPoints array (Double_t[NumberOfPoints][3]), which contains for each point the coordinates x,y,z, which correspond the indices 0,1,2.

ErrorPoints array (Double_t[NumberOfPoints][3]), which contains the corresponding errors

Maximum array (Double_t[NumberOfPoints]), which contains the maximum pulse hight of the point

MaximumError error on Maximum (Double_t[NumberOfPoints])

ChargeShare array (Short_t[NumberOfPoints]) is the number of pads which are used to calculate the hit coordinate. For a value > 1 there is charge sharing

In this module the *mapping* between channels and pads is applied. Further information can be found in chapter 3.1.1.

DoubleFit provides different methods for extracting the coordinates from the pulse information. For all presented methods the following cuts are applied.

- the event must contain more than `CutChannels` pulses
- the found pulses or clusters must be larger than `MinMaximum`, otherwise they are not stored. At a first stage all pulses are considered, because they could be tails of other pulses
- in the case that only points shall be stored whose coordinates are calculated with the information of more than one pad, `ChargeShare` must be activated

The method GaussZMeanX provides a Gaussian fit in z for each pad. After that the information of all pads which belong to one pulse is combined. This method is still included in the program code although it does not work in a satisfying way.

The two remaining methods will be explained in the following.

CenterOfGravity

This method uses all pads of one row at the same time (see Fig. 2.2). The pusles on

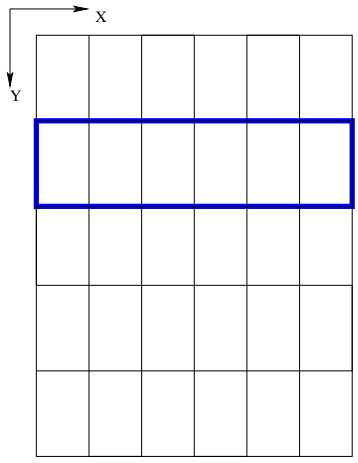


Figure 2.2: Center of Gravity arbeitet zeilenweise

the pads are filled in a two dimensional histogram. In this histogram the biggest entry is searched for. The corresponding bin is the starting point of the cluster search which searches for all neighboring bins belonging to the cluster, based on the depth-first search.

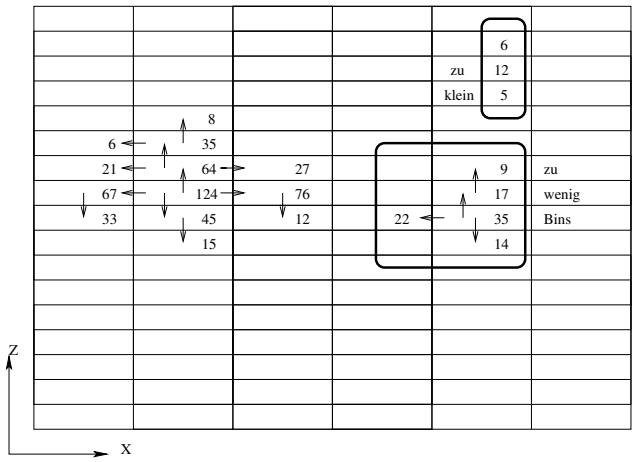


Figure 2.3: Approach of the cluster algorithm based on the depth-first search.

(see Fig. 2.3). Bins are only added to the cluster if they are smaller than the preceeding one. `COGSignificant` determines the size of this difference, in units of the bin error. They are are determined by the Poissonian distribution of the primary elctrons. The converting factor from ADC counts to primary electrons can be adjusted with the parameter `ElectronsPerADC`, which is set in the config file. Negative values mean that the added bin is allowed to be larger, and the value 0 represents "lower equal". With this approach closely together lying clusters are separated. A disadvantage of this approach is, that a bin which may belong to several clusters is counted as belonging to one single cluster.

The center-of-gravity of the clusters is then calculated using the weighted mean. The error on this quantity is determined by the errors on the individual weights, i.e. the bin contents. If only one pad is hit the error is set to $(padwidth)/\sqrt{12}$.

A point is only stored if the corresponding cluster consists of a minimum number of pads. This number is denoted `COGMinBinPerCluster`. But in any case the corresponding bins are removed from the histogram.

If the maximum found is lower than `MinMaximum` the procedure is stopped. So it is assured that each cluster exceeds the minimum threshold.

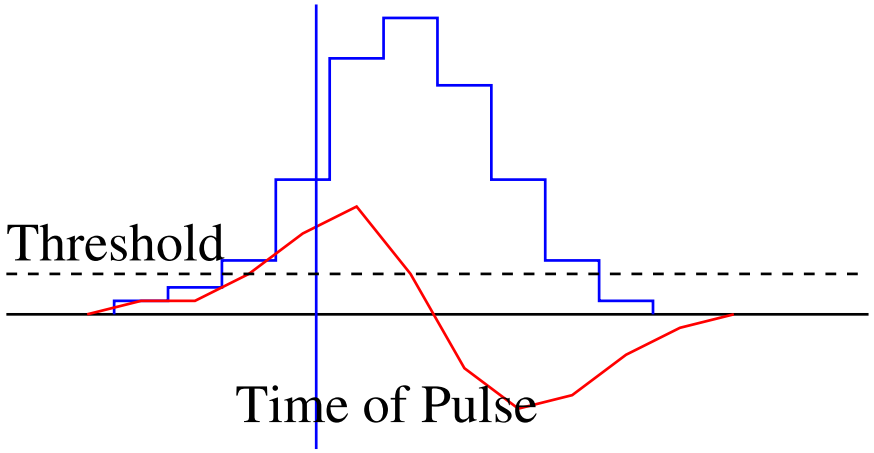
HitMerge

The method HitMerge uses a different approach for the separation of hits or clusters. First for all pads, arrays are created with the maximum number of time slices. These are filled with the pulses. This makes sure that two neighboring pulses can be treated in a reasonable way, even if they are stored in the raw data as two distinct pulses.

This array is now being read. If an entry is above the threshold `HMThresholdStart` a new hit is defined, which includes the preceeding bin, too (see Fig. 2.4). beginfigure[ht]

Figure 2.4: A hit, its corresponding derivative and time information.

.



The ending of a hit is either given by falling below the value `HMThresholdStop`, or by considering the differences of two succesive bins

$$D_i = ADC_i - ADC_{i-1}\,. \tag{2.1}$$

If the difference is below 0 and the following difference is greater than 0, this means that a rising edge will follow. In order to account for similar effects caused by noise, the difference of the two differences must exceed the value `fgNoiseValue`. In this case two hits are close together and must be separated. The actual bin is divided and the fraction

$$\frac{D_{i-1}}{D_{i-1} - D_i}$$

constitutes the last bin of the actual hit. The remaining bins are used for the next hit (see Fig.2.5).

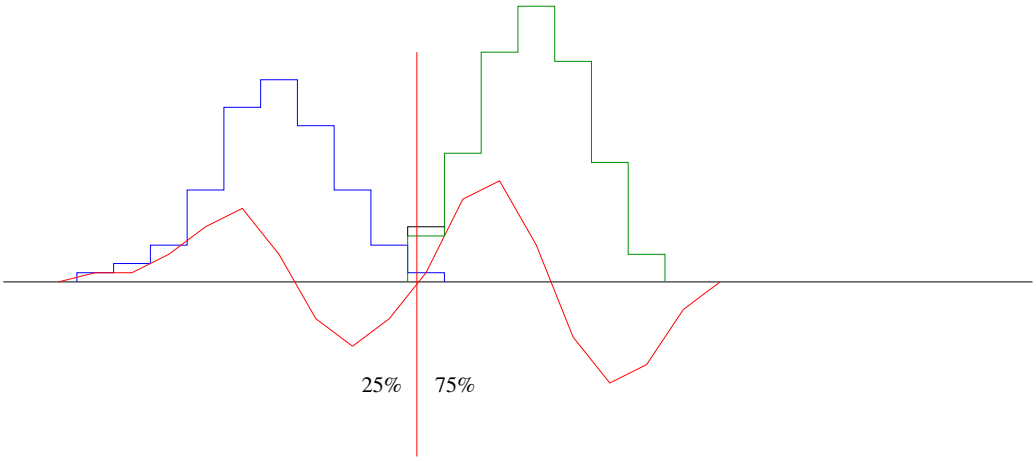


Figure 2.5: Separation of two hits with HitMerge

Once all hits are found, they are combined as follows. Beginning with the hit which contains the highest integrated charge a time window of the width 2×`HMTimeWindowWidth`

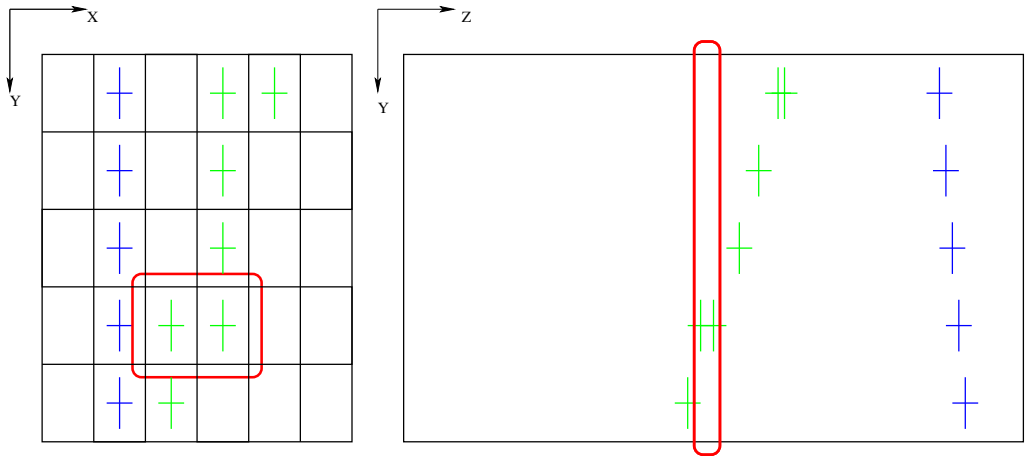


Figure 2.6: HitMerge combines the hits which lie within the time window.

is defined (in units of time slices). In this window a search for associated hits is performed (see Fig. 2.6). This is done by searching for hits in the xy plane, which lie within the distance which is defined by `HMMaxSearchRadius` (in units of mm). If another hit is found, it is searched again with this hit as starting point, whereas the time window remains unchanged.

At the moment the hit distance in the y coordinate is enlarged artificially, which means that hits lying in the same row are combined only.

2.2.2 TrackFinder

The points calculated by the Cluster Finder are combined to tracks by the module TrackFinder. At the moment the so called track-following-algorithm is implemented only. This algorithm is initialized with two points. This track hypothesis is used in the search for a further point. If an additional point is found it is added to the track and the parameters of the fit are calculated again. The new track hypothesis is used to perform a new search. The new track point is flagged and will not be used for the hypothesis of another track.

The program assumes that a track runs from top to bottom. This asumption is already used in the cluster finding. In order to run the program on data from a laser with horizontal beam, the chamber must be rotated virtually using the mapping.

The algorithm works row wise. This means that only one point per row can belong to a track. For the construction of the first trackhypothesis two rows are chosen which have a minimum distance of `MinSecondYLayer` and a maximum distance of `MaxSecondYLayer`. The first row is searched for starting from the bottom, whereas the second row can be any of the following ones allowed by the above mentioned conditions. Every combination of unused points belonging to one of the two rows is used as starting hypothesis.

In the xy as well as in the yz projection a χ^2 fit is performed. For the investigated row (from bottom to top) coordinates of the expected point are calculated. A box is then defined around this point with the dimensions 2·`FitAroundX`·`PadLengthX` ×2·`PadLengthY`×2·`FitAroundZ`· `TimesliceLength`[1]. Within this box it is searched for points which may belong to the track. If there exists more than one candidate, the one with the smallest χ^2 is chosen. If no candidate is found the search is stopped and it is checked if the found track shall be stored.

[1] TimesliceLength=TimeSliceWidth·DriftVelocity

A track is stored if the following conditions are fulfilled.

- it contains at least `MinPoints` reconstructed points
- χ^2 probability of the fit $\leq$`FitProb`

The file created by the TrackFinder module contains the following new information.

NumberOfTracks number of tracks stored

NumberOfPointsPerTrack number of points/track (Int_t[NumberOfTracks])

TrackCollection coordinates of the points belonging to the track in the array DataPoints (Int_t[NumberOfTracks][100]), it is only written and read out up to `NumberOfPointsPerTrack`

TrackChargeShare number of points of the track with charge sharing, i.e. number of points for which ChargeShare> 1 (Short_t[NumberOfTracks])

2.2.3 TrackFitter

The TrackFitter module performes a straight line χ^2 fit using the points calculated by the TrackFinder module. In addition, residuals and further statistical quantities are calculated and stored:

BreakThroughPointX x coordinate of the passing point of the reconstructed track (Double_t[NumberOfTracks])

BreakThroughPointZ z coordinate of the passing point of the reconstructed track (Double_t[NumberOfTracks])

SlopeX slope of the fitted straight line in the xy projection (Double_t[NumberOfTracks])

SlopeZ slope of the fitted straight line in the zy projection (Double_t[NumberOfTracks])

ErrorBreakThroughPointX error on the quantity (Double_t[NumberOfTracks])

ErrorBreakThroughPointZ error on the quantity (Double_t[NumberOfTracks])

ErrorSlopeX error on the quantity (Double_t[NumberOfTracks])

ErrorSlopeZ error on the quantity (Double_t[NumberOfTracks])

Chi2X χ^2 of the straight line fit in xy (Double_t[NumberOfTracks])

Chi2XPerN χ^2 per degrees of freedom of the fit in xy (Double_t[NumberOfTracks])

ProbX fit probability in x (Double_t[NumberOfTracks])

Chi2Z χ^2 of the straight line fit in zy (Double_t[NumberOfTracks])

Chi2ZPerN chi^2 per degrees of freedom of the fit in zy (Double_t[NumberOfTracks])

ProbZ fit probability in z (Double_t[NumberOfTracks])

Chi2total $\chi^2_x + \chi^2_z$ (Double_t[NumberOfTracks])

Direction normalized vector of the track (Double_t[NumberOfTracks][3])

DirectionError error on the track (Double_t[NumberOfTracks][3])

MinDistance minimum distance in the chamber between two tracks in one event. It is assigned the value of the diagonal distance of the chamber if only one track is found in the event.

MinPointDistance minimum distance of two points which belong to different tracks. It is assigned the length of the diagonal distance of the chamber, if only one track is found in the event.

DistanceInXatY distance of the point with respect to the track: the track is fitted again without the specific point and then the distance is determined. The result is stored in an array depending on the distance. In addition to the rows, the chamber is devided into 16 different subdivisions of the drift length. Here, MaxZMm [2] is devided into intervals of the same size. The position in the array is determined as follows: Double_t[Tracks][Zeile][Intervall] (Double_t[NumberOfTracks][20][20])

DistanceInZatY the same quantity for the distance in z (Double_t[NumberOfTracks][20][20])

DistanceInXatYlocal $\frac{x_1-x_3}{2-x_2}$; here, x_i belongs to to the 3 rows lying among one another. The storing is performed as in the variable DistanceInXatY (Double_t[NumberOfTracks][20][20])

DistanceInZatYlocal the same for the z coordinate: $x \rightarrow z$ (Double_t[NumberOfTracks][20][20])

[2] MaxZMm= TimeSliceWidth·DriftVelocity·BinsZ

Chapter 3

Usage

The program is started with

`DoubleFit [configfile]`

If no configuration file is specified, the program uses automatically the file `DoubleFit.conf` als config file.

3.1 The Configuration File

The config file is an ASCII text file, where control variables can be set with

`variable = value # comment`

"value" can also be a list of more than one variable (e.g. `InputDataFiles`). This list can be continued in the following line using \. Important: after \ nothing but "Return" is allowed [...].

In the following an example is given (LCIO input is commented out)

```
#MakeReal = 0                          # 0 Bool
#SingleFileMode = 0                    # 0 Bool
#OnlineMode=0                          # 0 Bool
ClusterFinding = 1                     # 1 Bool
TrackFinding = 1                       # 0 Bool
TrackFitting = 0                       # 0 Bool
ProgramDisplay = 1                     # 0 Bool
ScreenShotMode = 1                     # 0 Bool
VerboseLevel = 2                       # 1 Int

InputFormat = ROOT                     # String
CFTreeName = h5001                     # "h5000" String
InputDataFiles = d20040109_00.root \
                 d20040109_01.root \
                 d20040109_02.root \
                                       # list of strings

#InputFormat = LCIO                    # String
#InputDataFiles = desytpc.20040109.000.slcio \
#                 desytpc.20040109.001.slcio
```

```
OutputFileSuffix = _20040109_000      # "" String

###Padgeometry
BinsX = 6                             # Int
BinsY = 5                             # Int
BinsZ = 512                           # Int
PadWidth = 2                          # mm Double
PadHeight = 3.8                       # mm Double
#PadLengthX = ?                       # mm PadWidth+PadDistanceX/2. Double
#PadLengthY = ?                       # mm PadHeight+PadDistanceY/2 Double
PadDistanceX = 0.2                    # mm Double
PadDistanceY = 0.2                    # mm Double
#MaxChannel = 64                      # 64 Int
PadMapping     = 0 5 10 15 59 54
                 1 6 11 63 58 53
                 2 7 12 62 57 52
                 3 8 13 61 56 -1
                 4 9 14 60 55 50
                                      # list of Int

Calibration = 0.828 0.673 0.734 0.590 0.717 0.705\
              0.781 0.703 0.685 0.703 0.689 0.647\
              0.756 0.754 0.633 0.678 0.747 0.916\
              0.842 0.663 0.696 0.801 0.743 0.803\
              0.998 0.901 1.054 1.073 0.089 1.274
                                      # nX1. list of Double

#### DAQ
#TimeSliceWidth = 80                  # 80 ns Double
#### Gas
#DriftVelocity = 3.9                  # 3.9 cm/mus Double
#ElectronsPerADC = 120./670.          # 120./670. Double


#### Cluster finding
MinMaximum = 10                       # 12 ADC-Counts Double
#CutChannels = 5                      # 5 Pulse Int
#ChargeShare = 0                      # 0 Bool

UseGaussZMeanX = 0                    # 0 Bool

UseCenterOfGravity = 0                # 1 Bool
COGSignificant = -0.4                 # 0.0 sigma Double
COGMinBinPerCluster = 6               # 6 ADC-bins Int
COGNoiseValue=3.                      # 0. ADC-Counts Double

UseHitMerge=1                         # 0 Bool
HMTimeWindowWidth=1.2                 # 1.2 ADC-slicess Double
#HMMaxSearchRadius=.                  # mm PadLengthX+1. Double
```

```
HMThresholdStart=8.                    # 10.0 ADC-Counts Double
HMThresholdStop=8.                     # ADC-Counts HMThresholdStart Double
HMNoiseValue=3.                        # 4.0 ADC-Counts Double

#DoZCut = 1                            # 0 Bool
#ZCutAddLayers = 128                   # Int
#ZCutPersent = 0.575                   # Double
#ZCutBorderActivity = 42               # Int
#ZCutBorderActivityCenter = 0          # Int


### Track finding
#FitAroundX = 0.2                      # *PadLengthX  0.2 Double
#FitAroundZ = 3.5                      # *TimesliceLength 3.5 Double
MaxSecondYLayer = 13                   # MaxYBin Int
MinSecondYLayer = 7                    # 4 Int
MaxMissedPoints = 6                    # 2 Int
#FitProb = 0.                          # 0.0 Double
MinPoints = 6                          # MaxYBin-2 Int
```

All control variables and the corresponding default values are shown in Table 3.1 to Table 3.4.

Table 3.1: Variablen des Config-Files : Programmsteuerung

Variable	Typ	Defaultwert
MakeReal	Bool	false
SingleFileMode	Bool	false
ClusterFinding =	Bool	true
TrackFinding	Bool	true
TrackFitting	Bool	true
OnlineMode	Bool	false
ProgramDisplay	Bool	false
ScreenShotMode	Bool	false
VerboseLevel	Int	1
InputFormat	String	-
InputDataFiles	String[]	-
OutputFileSuffix	String[]	""
CFTreeName	String[]	h5000
OnlineMode	Bool	false

3.1.1 Pad↔Channel Mapping

Unfortunately the channels are not always corrected to the pads in the right order. In this case the right mapping is reconstructed by using software. In order to do so, an array Int_t[MaxPads] serves as input to the configuration file. This array contains for each pad the corresponding channel number. Pads which are not connected are flagged with -1. Both pad and channel numbers start from 0 (see Fig. 3.1). For reasons of clearness it is

Table 3.2: Variablen des Config-Files : Geometrie

Variable	Typ	Defaultwert	Einheit
BinsX	Int	-	
BinsY	Int	-	
BinsZ	Int	-	
PadWidth	Int	-	mm
PadHeight	Int	-	mm
PadDistanceX	Int	-	mm
PadDistanceY	Int	-	mm
MaxChannel	Int	64	
PadMapping	Int[]	-	
TimeSliceWidth	Double	80	ns
DriftVelocity	Double	3.5	cm/μs
ElectronsPerADC	Double	120./670.	

Table 3.3: Variablen des Config-Files : Cluster Finding

Variable	Typ	Defaultwert	Einheit
MinMaximum	Double	12	ADCcount
CutChannels	Int	5	
ChargeShare	Bool	false	
UseGaussZMeanX	Bool	false	
UseCenterOfGravity	Bool	true	
COGSignificant	Double	0.	σ
COGMinBinPerCluster	Int	6	ADCbin
UseHitMerge	Bool	false	
HMTimeWindowWidth	Double	1.2	ADCslice
HMMaxSearchRadius	Double	PadLengthX+1.	mm
HMThresholdStart	Double	10.	ADCcount
HMThresholdStop	Double	HMThresholdStart	ADCcount
HMNoiseValue	Double	4	ADCcount

Table 3.4: Variablen des Config-Files :Track Finding

Variable	Typ	Defaultwert	Einheit
FitAroundX	Double	0.2	·PadLengthX
FitAroundY	Double	0.5	·PadLengthY
FitAroundZ	Double	3.5	·TimesliceLength
MaxSecondYLayer	Int	MaxYBin	
MinSecondYLayer	Int	4	
MaxMissedPoints	Int	2	
FitProb	Double	0	
MinPoints	Int	MaxYBin-2	

0 0	1 5	2 10	3 15	4 59	5 54
6 1	7 6	8 11	9 63	10 58	11 53
12 2	13 7	14 12	15 62	16 57	17 52
18 3	19 8	20 13	21 61	22 56	23 −1
24 4	25 9	26 14	27 60	28 55	29 50

Figure 3.1: Pad (upper right)↔channel (lower left) mapping

recommended to enter the array according to its geometry. The example shown is used for the TPC operation with a horizontal laser beam. Therefore the chamber is rotated virtually by $\frac{\pi}{2}$ dreht. Additionally it is assumed that the connector of the channels 48-63 is mounted in the wrong direction. Channel 51 is broken.

3.2 The Display Function

A ROOT canvas is created, which shows the process flow.

3.2.1 Screenshotmode

If this mode is activated, the input of a character is expected after each change of the program display. If only “Return” is entered, the program proceeds up to the next change. If “p” is entered, a postscript file is created, and the program continues.

Bibliography

[1] *LCIO home page.* `http://www-it.desy.de/physics/projects/simsoft/lcio/.`