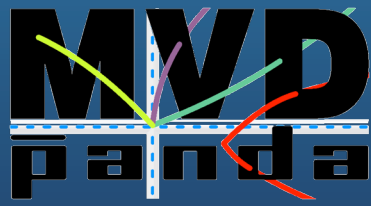
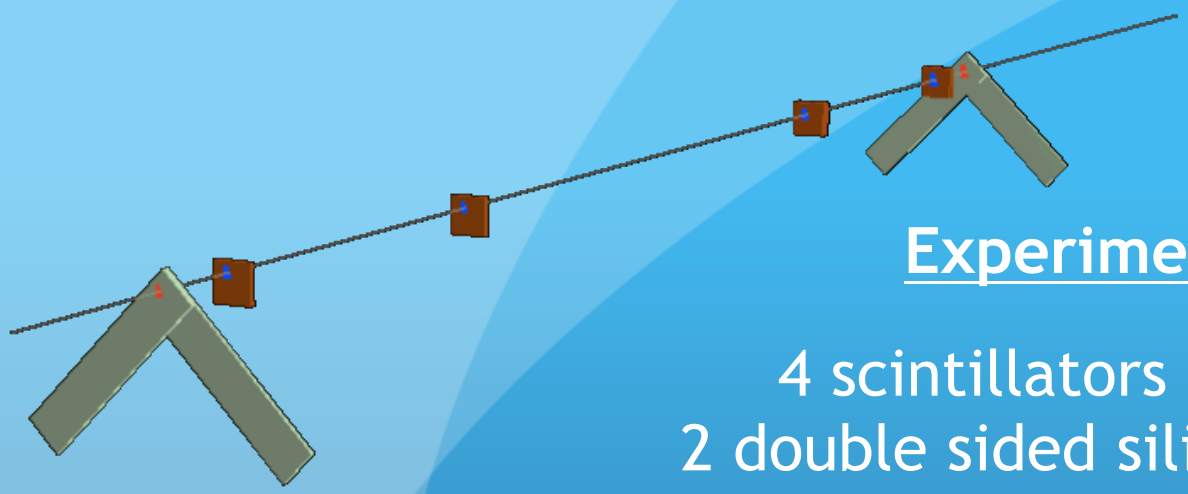
Bonn-Cologne Graduate School
of Physics and Astronomy

Bonn Test Station data analysis with PandaRoot

Simone Bianco

universitätbonn





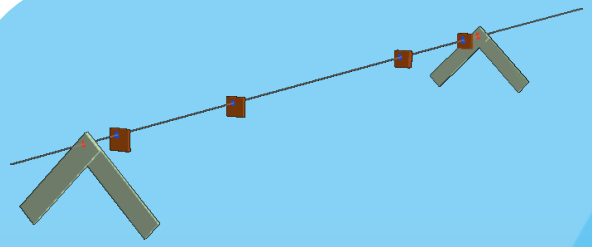
Experimental Setup:

4 scintillators (for the trigger)
2 double sided silicon strip detectors
4 single sided silicon strip detector

Ouput of the DAQ:

Text file containing information about the hits: FE module, strip number, ADC counts, lengths of the signal (in clock cycles)

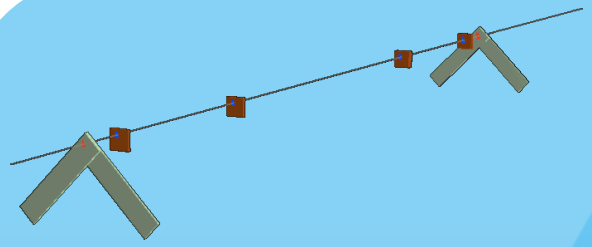
Sizes Sensors	1.92 cm x 1.92 cm
Thickness	300 μm
Pitch	50 μm



Real World (R.W.): In the DAQ the frontend modules are numbered consecutively from 0 to the total number of FE -1.

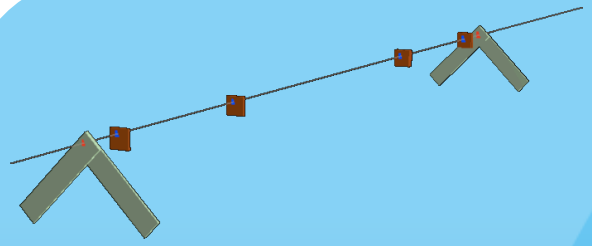
Software World (S.W.): In the mvd code in pandaroot FE modules do have numbers between 0 and the number of FFE of one sensor. For each sensor the numbering starts from zero.

We want to plug the real data in the framework at the point between the digitization and the reconstruction, so we must reproduce the PndMvdDigiStrip standards.



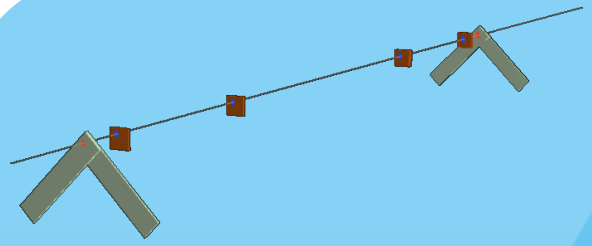
TOOLS:

Geometry Generator	**This is a set of macros creating geometry matching the real setup. The same files are used to simulate the experiment**
FE Mapper	This tool is reading the geometry listing in a file the volumes and the corresponding S.W. frontend IDs. One can edit this file to change the default mapping of the R.W. frontends.
Converter	At this step the map is read, together with a calibration file (ADC counts → charge) and the real data. The output is a file containing PndMvdDigiStrip.
Reconstruction	Here the previous file is analyzed and the standard MVD reconstruction is performed. The detector name is set correctly by the Converter, so the reco can select the right parameters.



Mapper

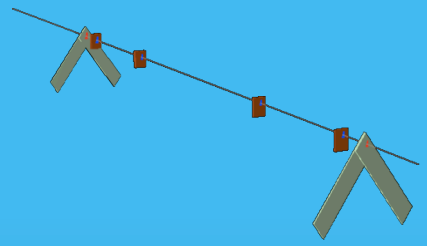
```
FairRuntimeDb* rtdb = fRun->GetRuntimeDb();

FairParAsciiFileIo* parInput = new FairParAsciiFileIo();
parInput->open(digiparFile.Data(),"in");
rtdb->setFirstInput(parInput);

FairParRootFileIo* output=new FairParRootFileIo(kTRUE);
output->open(parFile.Data());
rtdb->setOutput(output);

fRun->SetGeomFile(geomFile); // set filname
fRun->LoadGeometry();

PndMvdCreateDefaultApvMap* creator = new PndMvdCreateDefaultApvMap();
fRun->AddTask(creator);

fRun->Init();
```



Template of a map

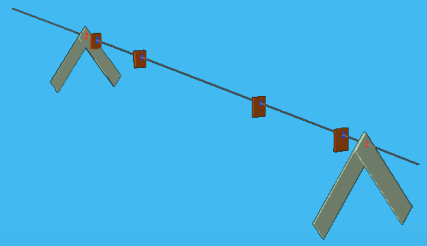
RW	SW	Detector Name
0	0	/TS_1/TTVol_0/TTDouble_0/StripActiveTD1_0
1	1	/TS_1/TTVol_0/TTDouble_0/StripActiveTD1_0
2	2	/TS_1/TTVol_0/TTDouble_0/StripActiveTD1_0
3	3	/TS_1/TTVol_0/TTDouble_0/StripActiveTD1_0
4	4	/TS_1/TTVol_0/TTDouble_0/StripActiveTD1_0
5	5	/TS_1/TTVol_0/TTDouble_0/StripActiveTD1_0
6	0	/TS_1/TTVol_0/TTSingle_0/StripActiveTS3a_0
7	1	/TS_1/TTVol_0/TTSingle_0/StripActiveTS3a_0
8	2	/TS_1/TTVol_0/TTSingle_0/StripActiveTS3a_0
-1	3	/TS_1/TTVol_0/TTSingle_0/StripActiveTS3a_0
9	0	/TS_1/TTVol_0/TTSingle_0/StripActiveTS3b_0
10	1	/TS_1/TTVol_0/TTSingle_0/StripActiveTS3b_0
11	2	/TS_1/TTVol_0/TTSingle_0/StripActiveTS3b_0
-2	3	/TS_1/TTVol_0/TTSingle_0/StripActiveTS3b_0
12	0	/TS_1/TTVol_0/TTSingle_0/StripActiveTS4a_0
13	1	/TS_1/TTVol_0/TTSingle_0/StripActiveTS4a_0
14	2	/TS_1/TTVol_0/TTSingle_0/StripActiveTS4a_0
-3	3	/TS_1/TTVol_0/TTSingle_0/StripActiveTS4a_0
15	0	/TS_1/TTVol_0/TTSingle_0/StripActiveTS4b_0
16	1	/TS_1/TTVol_0/TTSingle_0/StripActiveTS4b_0
17	2	/TS_1/TTVol_0/TTSingle_0/StripActiveTS4b_0
-4	3	/TS_1/TTVol_0/TTSingle_0/StripActiveTS4b_0
18	0	/TS_1/TTVol_0/TTDouble_0/StripActiveTD2_0
19	1	/TS_1/TTVol_0/TTDouble_0/StripActiveTD2_0
20	2	/TS_1/TTVol_0/TTDouble_0/StripActiveTD2_0
21	3	/TS_1/TTVol_0/TTDouble_0/StripActiveTD2_0
22	4	/TS_1/TTVol_0/TTDouble_0/StripActiveTD2_0
23	5	/TS_1/TTVol_0/TTDouble_0/StripActiveTD2_0

Converter



```
// -----  Parameter database   ----------------------------------------
  FairRuntimeDb* rtdb = fRun->GetRuntimeDb();

  FairParAsciiFileIo* parInput = new FairParAsciiFileIo();
  parInput->open(digiparFile.Data(),"in");
  rtdb->setFirstInput(parInput);

  FairParRootFileIo* output=new FairParRootFileIo(kTRUE);
  output->open(parFile.Data());
  rtdb->setOutput(output);

  fRun->SetGeomFile(geomFile); // set filname
  fRun->LoadGeometry(); // set the flag

  // -----  Converter  ----------------------------------------------------
  PndMvdConvertApv* ApvConverter= new PndMvdConvertApv(CalibFileName, HitFileName);
  PndMvdMapApv* ApvMapper = new PndMvdMapApv(MapFileName);

  PndMvdConvertApvTask* convertTask = new PndMvdConvertApvTask(ApvConverter,ApvMapper);
  convertTask->SetVerbose(3);
  fRun->AddTask(convertTask);

  fRun->Init();

  long int  nEvents = ApvConverter->GetNofEvents();
  cout<<" ---- Start RUN ----"<<endl;
  fRun->Run(0,nEvents);
```

Converter - Single Sided Modules

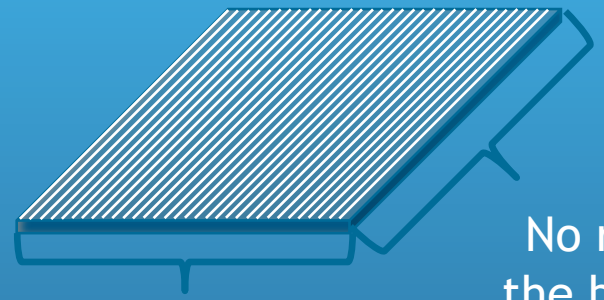


The converter is creating PndMvdDigiStrip objects.
The two single sided sensors (components of each of the single sided boxes) are treated independently.

The MVD strip reconstruction tools are designed to work with double sided sensors.

In the definition of the parameters we set one ideal strip on the bottom side, choosing as a pitch the width of the sensor.

When a single sided sensor is hit we fill the TClonesArray with one more digi on the bottom side:

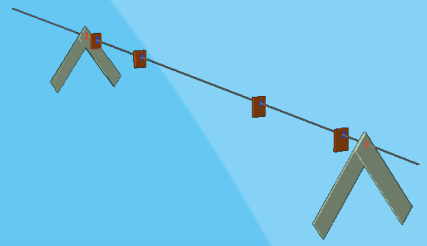


Top side:384 strips
3 Frontend chips

No readout of the bottom side

DetName, Index, …	Like hits on the top side
Charge	Sum of the values on the top side
Channel	0
Frontend chip	4

Converter - Calibration



The information about the collected charge in each strip is saved in terms of ADC counts.

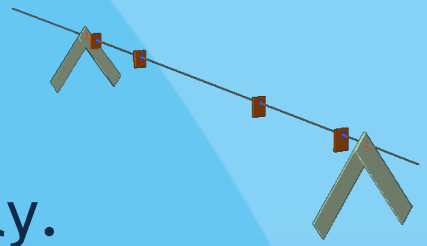
The charge calibration can realized in two ways:

1. the APV chips allow to inject charges at the frontend level, so we can map the number of ADC counts on each channel changing the injected charge;

we know nominal values of injected charges, but we are not absolutely sure about the exact values

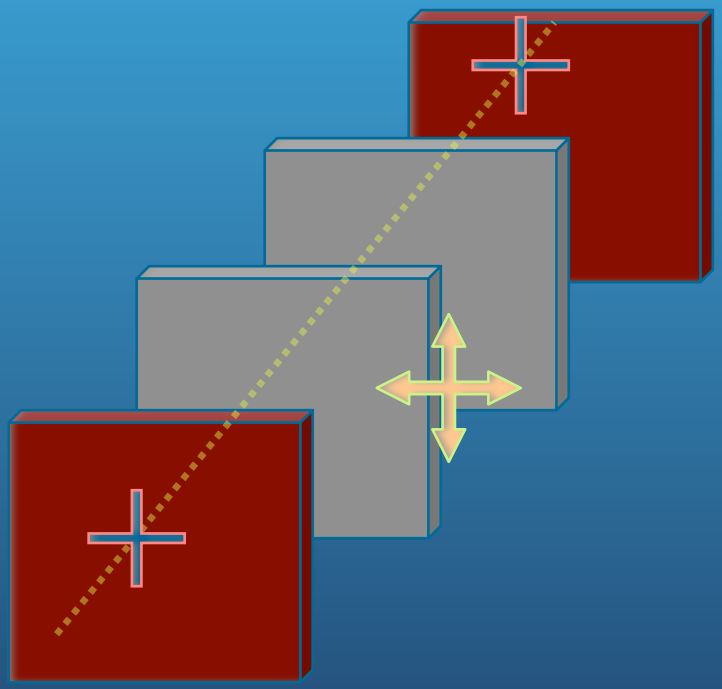
2. the protons of the beam are MIP, we can extract the absolute value from the peak of the Landau distribution filled in many events.

Alignment



The tracking station has been aligned optically.

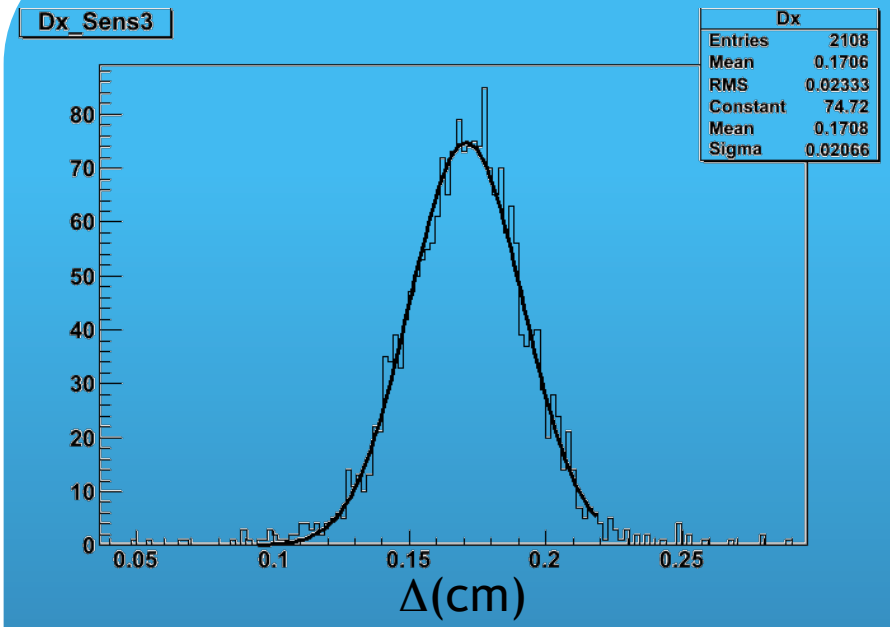
In the off-line analysis we perfom a further alignment using the following procedure:

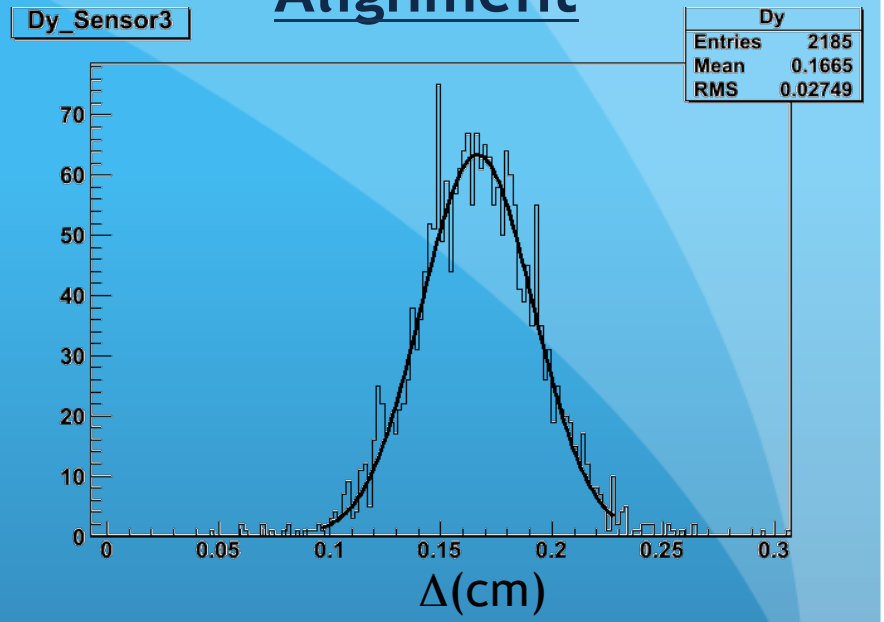


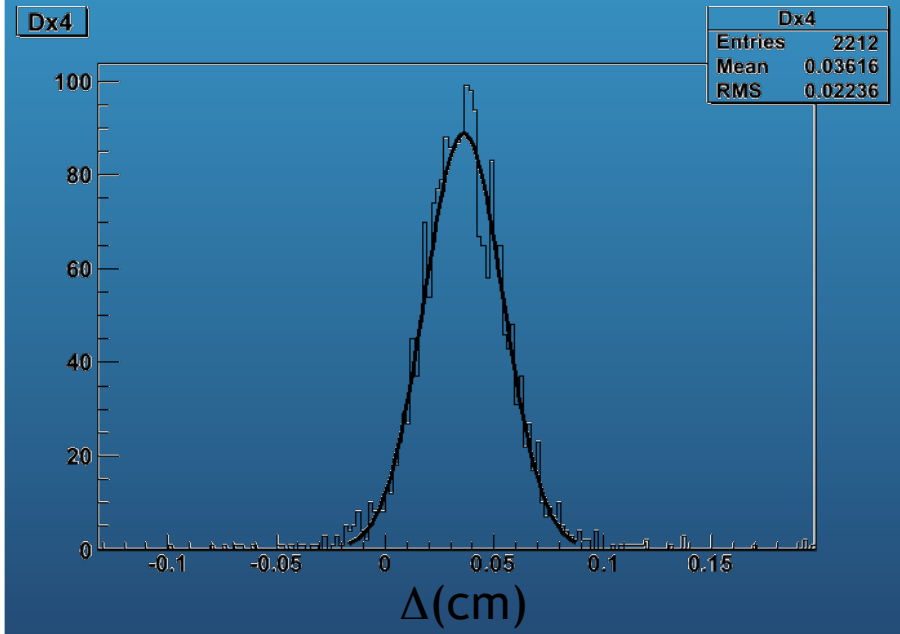
Hits on the first and last box are considered fixed. Via a straight line fit we calculate the residuals on the intermediate sensors.

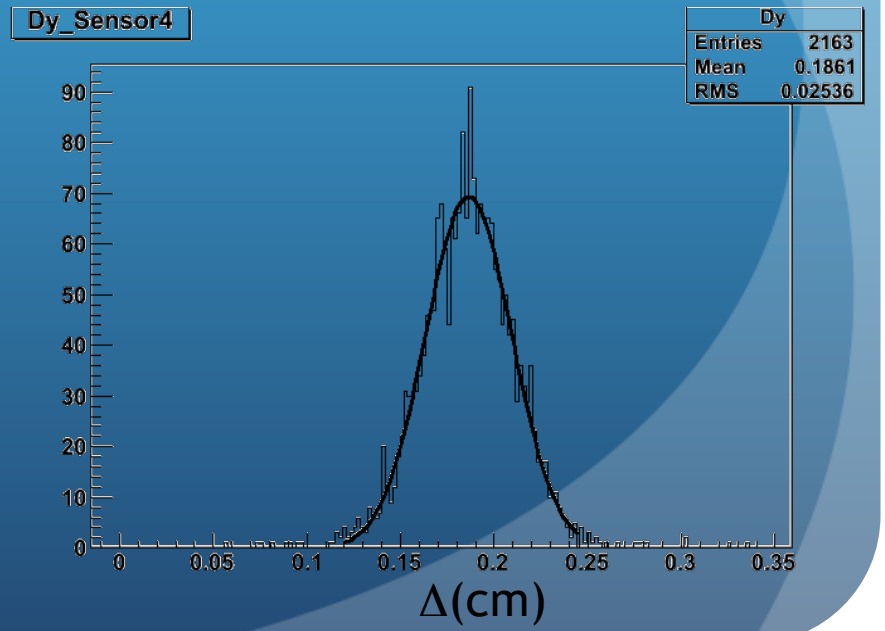
Offsets of the intermediate sensors

Alignment

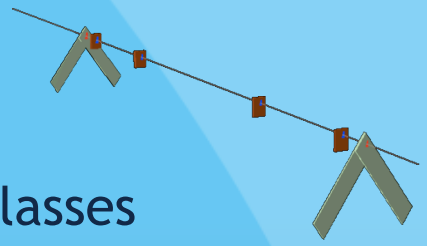

Dx_Sens3
Dx
Entries 2108
Mean 0.1706
RMS 0.02333
Constant 74.72
Mean 0.1708
Sigma 0.02066
Δ(cm)


Dy_Sensor3
Dy
Entries 2185
Mean 0.1665
RMS 0.02749
Δ(cm)


Dx4
Dx4
Entries 2212
Mean 0.03616
RMS 0.02236
Δ(cm)


Dy_Sensor4
Dy
Entries 2163
Mean 0.1861
RMS 0.02536
Δ(cm)

Reconstruction and Analysis

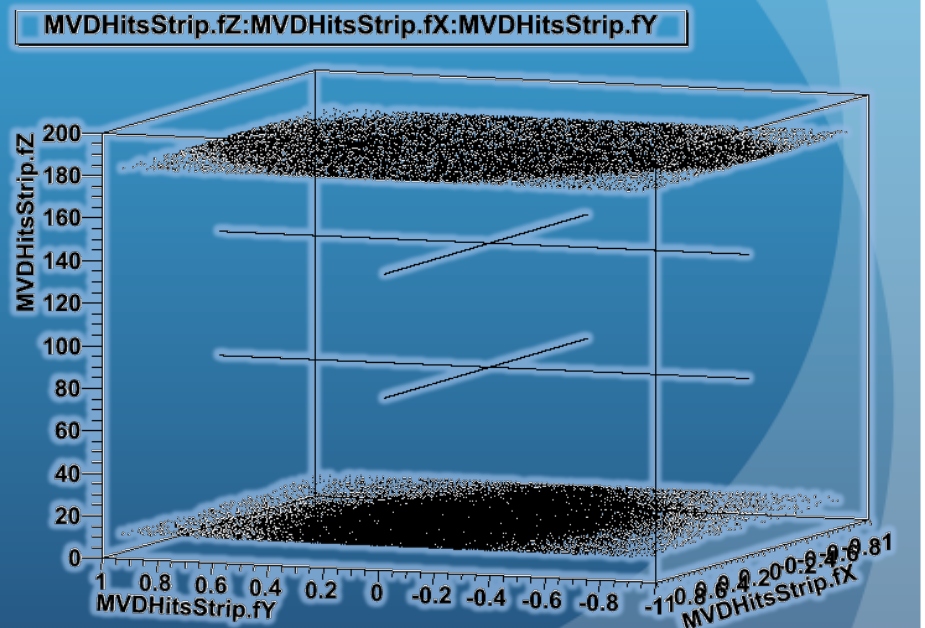


The reconstruction is performed using the MvdReco classes loading the converted digis.

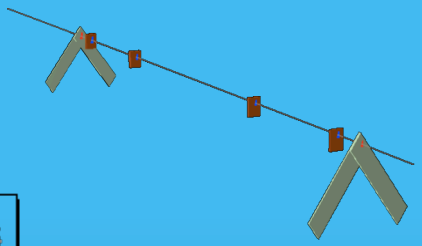
The reconstructing macro is reading the parameters from the ASCII file (working stored them in the .root parameters file during the conversion)

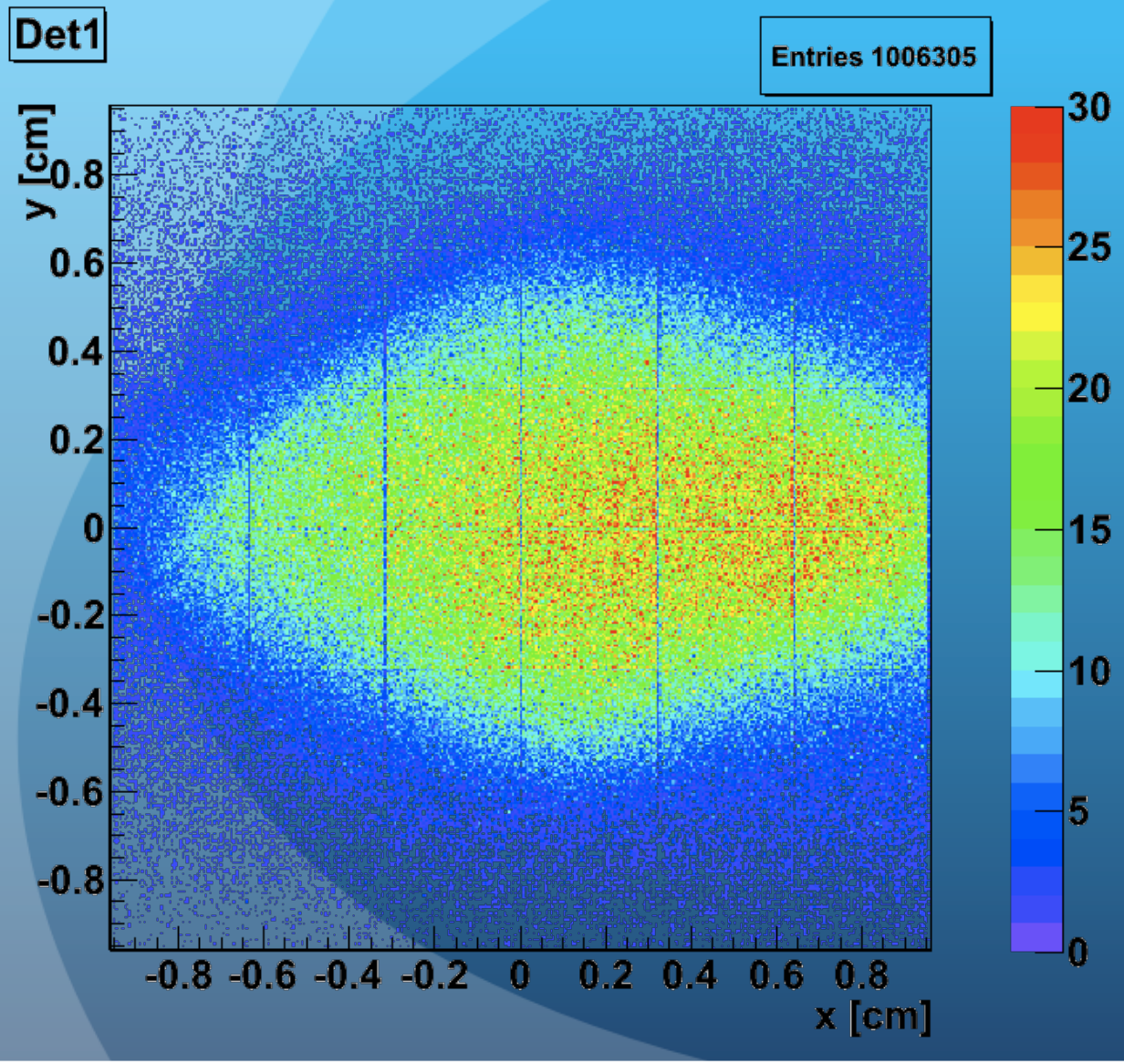
1-D information from the single sided sensors. When analyzing the data we can merge the data coming from the same box if there are no ambiguities (1 hit on each box).

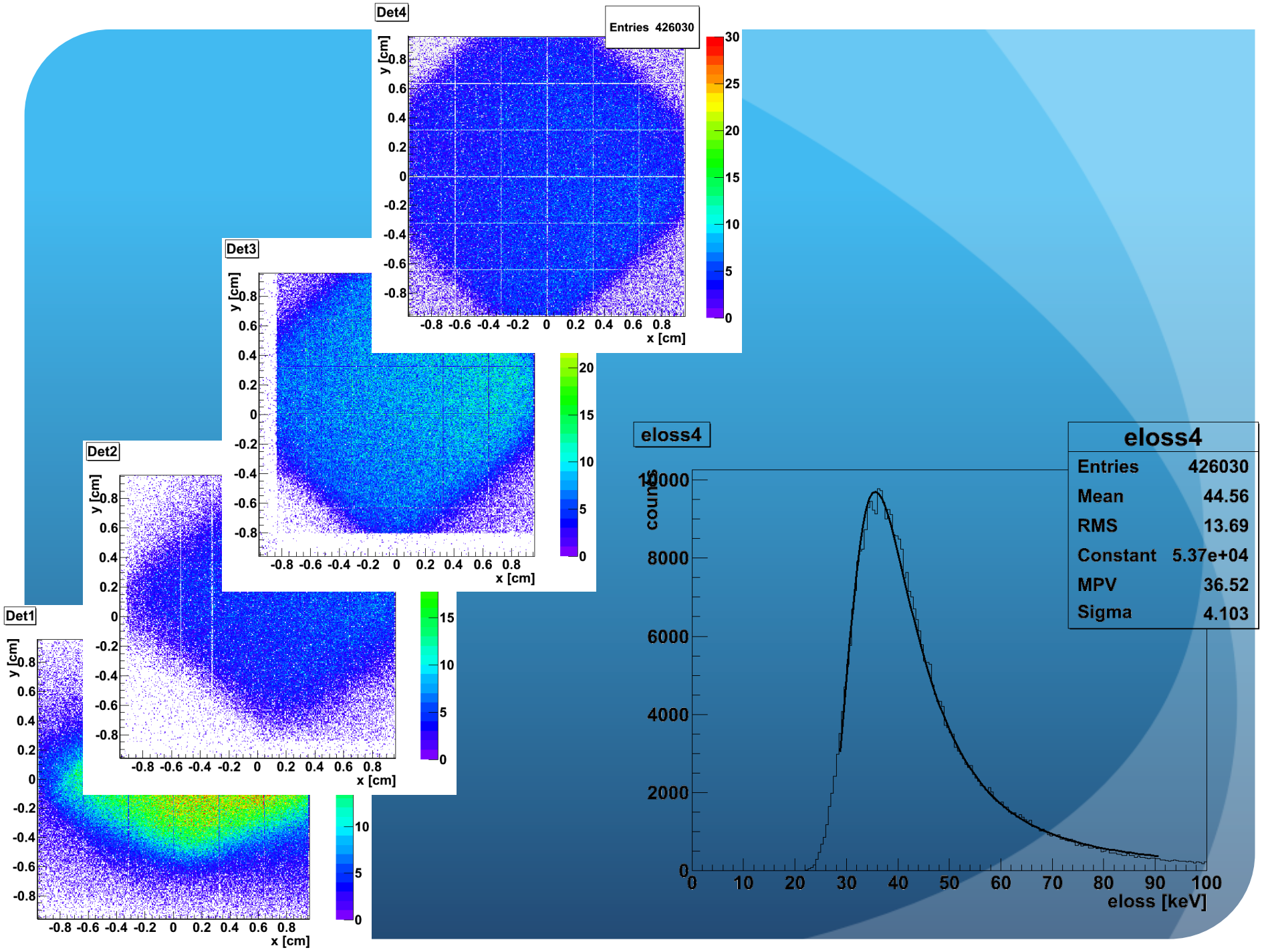
During this run we had a negligible number of events with more than one cluster/ sensor

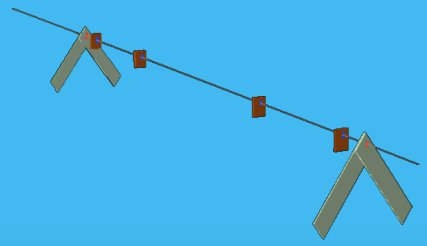


3 GeV/c protons at COSY






Det4
Entries 426030
y [cm]
x [cm]
Det3
Det2
Det1
eloss4
counts
eloss [keV]
eloss4
Entries 426030
Mean 44.56
RMS 13.69
Constant 5.37e+04
MPV 36.52
Sigma 4.103



Classes:

trunk/mvd/MvdDAQ/

Macros:

trunk/macros/mvd/BonnTS/

In the SVN trunk repository

THANKS FOR YOUR ATTENTION!