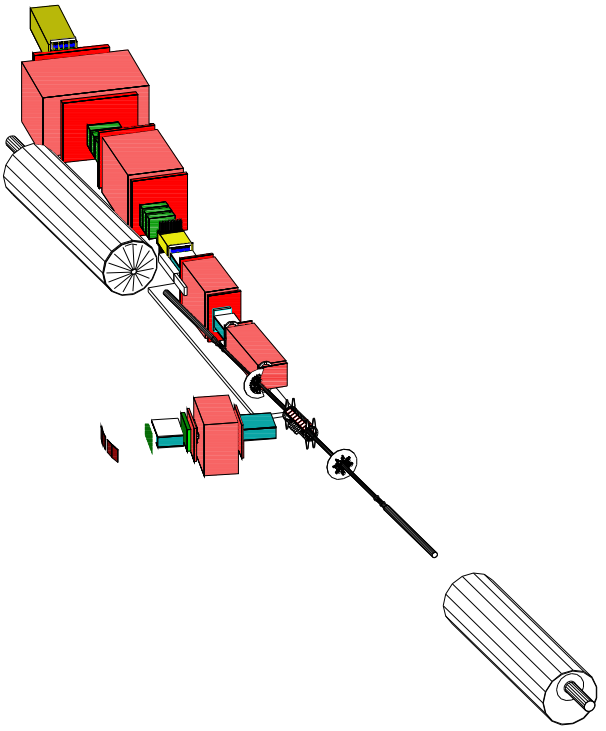
Guide for Data Reconstruction and Calibration

Djamel Ouerdane
Niels Bohr Institute, Copenhagen
ouerdane@nbi.dk

April 16, 2002

Abstract

This humble guide is a summary of what I did for data reconstruction and calibration. It gives some hints and explanations on calibration procedure and data reconstruction. It covers mainly how I deal with global tracking, BB counters and hodoscope calibrations.



Contents

List of Figures

1 Introduction

This guide is meant to help any member of the BRAHMS collaboration with data reconstruction and offline detector calibration. The content deals mainly with what I personally implemented for that matter in terms of procedures, software and usage. This guide cannot obviously cover all aspects in every detail for two main reasons: the first one is simply that my technical knowledge covers certain parts of the experiment but certainly not all; the second one is that as I'm writing now, things change quickly (BRAT improves day after day and changes can be sometimes very drastic). Nevertheless, I think the core of this guide should remain fairly up to date within the next months.

Different topics are dicussed here. In order to make it easy for me and also for you, I'll introduce the following sections in the order I process the data from the raw format to a more useful one for physics analyses. Some of the steps will be mentioned quickly since I was not personally involved, but I'll indicate whom to contact for more details. Last but not least, there's an appendix at the end of the guide with hints and tricks about database connection, data reconstruction jobs, etc.

2 First Calibrations

The first, easiest and most trivial thing to do is the *pedestal* calibration. This should be done *before anything else*. The way I do it requires that you can submit jobs as `bramreco`. You should therefore have some knowledge about `CRASH` [1]. I assume it is the case. Moreover, I'll assume that you know about `bratmain, job configuration scripts, database connections`. This is all very well described in [2]. Finally, all procedures described in this guide have been tested and are still being used on the rcas machines, which leads to the final assumption that you work on these mahchines.

2.1 Pedestal calibration

Pick up the configuration script `CrashPed.C` in the subdirectory `share/brat/scripts/calib/` of your brat installation directory. From wherever on any rcas machine, write the `jsf` file (or job summary files) of the desired pedestal run (you can browse the DAQ run viewer to quickly find pedestal runs, they are tagged `PH` and the web link is

```
http://pii3.brahms.bnl.gov/daq-cgi-bin/cgiTrigSummary.perl
```

Edit the script. Check the database connection section and change if needed (but the default parameters should be ok, cf. Appendix A for database parameters). Go now to the module section. The `trigger filter` should select only 7 and 8. Then come the pedestal modules for all the hodoscopes, BB counters and Cherenkov detectors. Feel free to add or remove whatever. Each of these modules has 2 very important methods:

1. `SetSaveAscii(Bool_t)`
2. `SetCalibFile(const Char_t*)`

If you want to save the calibration to temporary ascii files (highly recommended), the 1st method should always set `kTRUE`. The 2nd method should set the name of this ascii file.

Another method allows you to set an upper limit to the width of the evaluated pedestals. If you think that the photo-multiplier tubes (PMT) *should not* have a pedestal width bigger than this limit, the ones that turn out to be over this limit will be tagged as `kCalException` (= -1111 internally). They will be ignored in subsequent analyses.

You only need to scan a sequence (pedestal runs normally have more than one sequence). Check the histogram file. In Fig.1, you can see a typical calibration for the BB right array (run 2476).

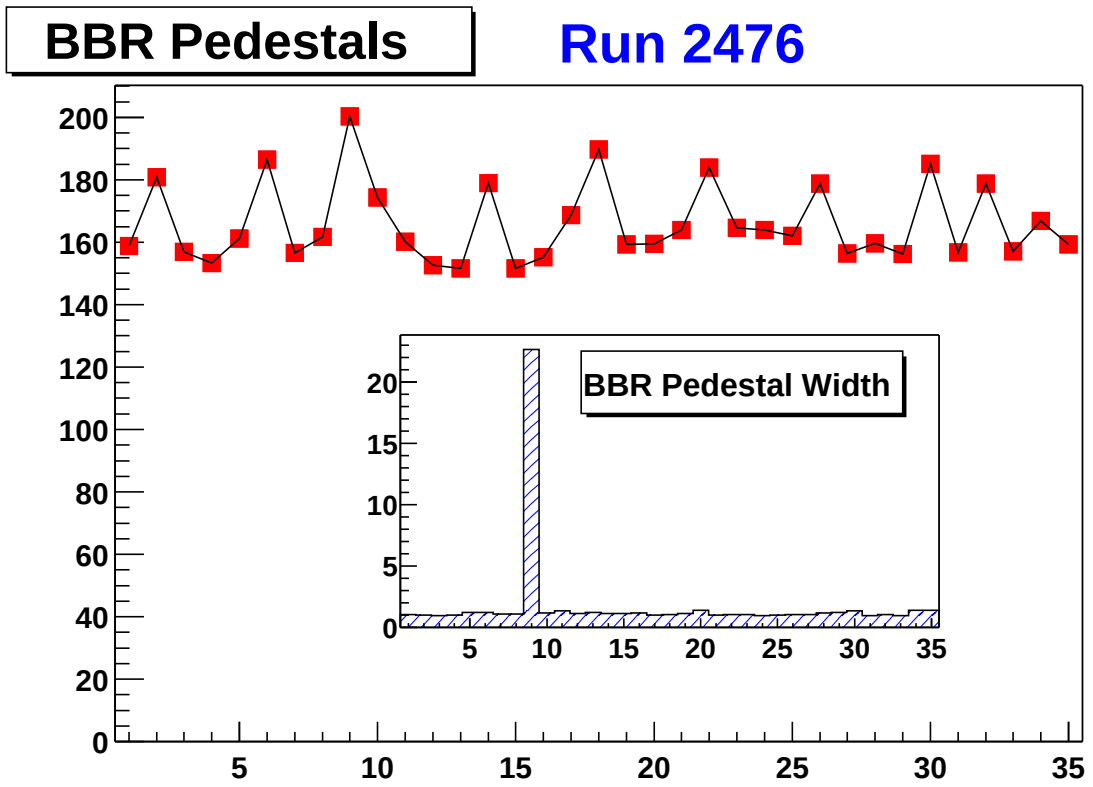


Figure 1: Pedestal calibration for BB right array.

The histogram file basically contains histogram directories named `<detector>_Pedestals`. In each of these directories, you will find summary histograms (pedestal values and widths versus PMT number) and subdirectories of histograms for individual PMT signal. Check them out. Correct the ascii files if you notice some weird behaviour in some histograms (bad fit, negative values, etc). You're now ready to commit this calibration to the database. This can be done from wherever on rcas. The bratmain script is `CommitPedCal.C`. Edit it and change if needed. Note that all calibration modules a have method `SetCommitAscii(Bool_t)`. It should be set to `kTRUE`. The ascii files you got previously will be read by the modules and committed to the database automatically. If you want to avoid any trouble, *don't change the format of these ascii files.* You're allowed to change numerical values only. If you have to set a kCalException for a given value, write -1111 and nothing else. Note also a method called `SetComment(const Char_t*)`. You

should write a comment string of at least 15 characters, describing wisely the calibration you achieved and want to check in.

A typical usage of the commit script would be:

```
bratmain CommitPedCal.C -r 2476 or
bratmain CommitPedCal.C -r 2476 -f 2502 or
bratmain CommitPedCal.C -r 2476 -f 2502 -M hilux12.nbi.dk -U ouerdane
```

If you want to share your calibration, I strongly suggest you keep the default database parameters.

2.2 Second Calibrations

In what follows, I work with data already reduced at the local tracking level. This data contains local tracks from all tracking chambers, raw digits for the other detectors (BB, TOF, Cherenkov, Mult arrays). The data files are located on some rcas directories, mainly `/brahms/data0<n>/igb/R<run>/seq/` with *<n>* = 1,2 and *<run>* is the run number. Ian Bearden is the one who reduced this huge amount of data.

If you want to know more about this reduction step, contact him at `bearden@nbi.dk`. Some TPC calibrations were also done by Peter Christiansen (drift velocity, time offsets, pad status), as well as some DC calibrations by Pawel Staszel. These calibrations are used for local tracking in the forward arm. Contact these guys if you want to know more about it (`pchristi@nbi.dk` and `staszel@nbi.dk`).

2.2.1 Beam-Beam counter calibration

Before any data reconstruction, you have to make sure that the Beam-Beam counters are calibrated for the data to be reconstructed. We have already performed a pedestal calibration for the BB arrays. For details about the BB, one can still read [3]. We have now to do more advanced calibrations in this order:

1. `TDC Gain`
2. `ADC Gain`
3. Δ`TDC` (delay between reference tube and other tubes)
4. `Slewing Correction` (refinement of ΔTDC)
5. `Vertex Offset` (offset along the beam line with TPM1 tracks)

All of these calibrations use the same machinery: a script that processes the data, writes histograms, fits those and saves the results to temporary ascii files, and a script to commit the calibration to the database. Since all calibration modules derive from a base class, they all have common methods (`SetSaveAscii`, `SetCommitAscii`, `SetLoadAscii`, `SetComment`, `SetCalibFile`.)

For each of these calibrations, a script can be found in

`<brat-install-dir>/share/brat/scripts/calib/bb`

with explicit names.

TDC Gain Calibration

This calibration is extremely important since it is what will allow us to convert TDC numbers to real time (in ns). This must be as precise as possible in order to get a good time resolution. Since there are small differences between TDC modules, one has to perform this calibration for each BB PMT and *not* apply blindly the nominal gain. But the `TDC Gain` calibration doesn't need to be done often. Once a year should be enough as far as I could check. But if you really want to do it, be sure to select a *TDC gain calibration run* for the data, otherwise you may have some bad surprises...Use the script `BbTdcGain.C` and `CommitTdcGain.C` like for the pedestals.

ADC Gain Calibration

The `ADC gain calibration` is important because you want to make sure that you select valid hits and not pedestal hits. Since each PMT has a different internal energy gain, the idea is to normalize the ADC signals by a common quantity, which is the ADC of a minimum ionizing particle (MIP). An ADC gain is much more unstable than a TDC gain due to high-voltage mean value shift. I would say that this should be done every week during a run period. To accumulate a reasonable statistics, you need to scan at least 15 sequences of a run. Make sure you select all triggers (1, 3, 4, 5, 6) to include peripheral events. The reason is that the 1st MIP peak should be the highest. If you select only central events, you may end up with multiple MIP peaks higher than the 1st one. Here is a typical usage of this script:

```
bratmain BbAdcGain.C -r 5662 -I <input dir> -H bbAdcGain5662.root
-v 5
```

Here, *<input dir>* refers to one of the `/brahms/data0n/igb`. You might want to set other options, etc. Feel free to add your own options. Note that there are also several ways to add sequence files to the input module, cf. Appendix B and [2].

Once the job is finished, open the histogram file and check histograms in `BBx_AdcGain` and `BBx_AdcGain/CalAdc`. In the first one, you have summary histograms with calibration values versus tube number. In the second directory, you have histograms for individual tubes after ADC gain normalization. In Fig.2, you can see a typical distribution. The 1st MIP peak should be centered at 1. So far, we haven't calibrated higher MIP peaks. I know by experience that a couple of tubes will show some weird distribution. If you're in doubt, ask experts (write to `brahms-l@bnl.gov`). If they don't know why they behave oddly, ignore these tubes and tag them as bad in the ascii file (-1111).

Like the pedestals, use `CommitAdcGain.C` to commit the calibration ascii file. You can now proceed with the Δ`TDC` step.

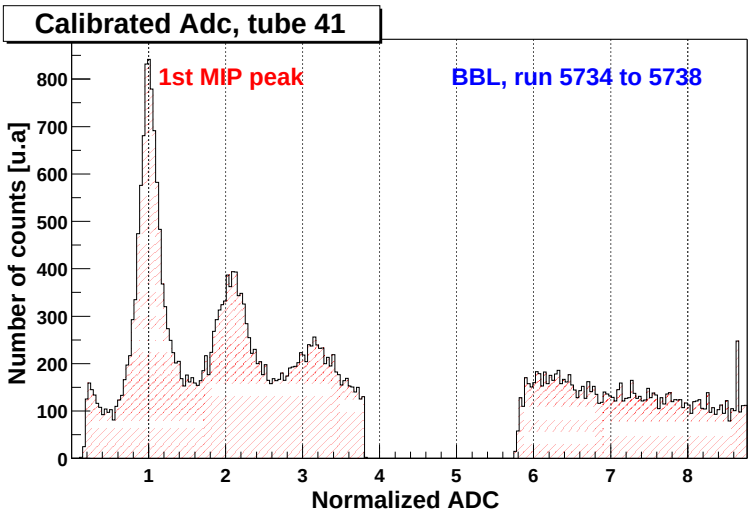


Figure 2: Single tube ADC distribution after gain calibration.

ΔTDC Calibration

This calibration is crucial for our vertex and start time determination. It will also contribute to the overall time of flight resolution so this calibration *must be good*. The idea behind is to align all tubes to a reference tube in time because there are some delays between them. These delays are due to e.g. different cable length. Since these tubes detect on average particles with the same properties (not completely true but a good assumption), their time distribution should be similar. That's why we should correct for the delay between them.

Use the script `BbDeltaTdc.C` and process the data in the same way. You can check the parameters for the module `BrBbDeltaTdcCalModule` to work properly. Four methods are of importance: `SetRef(Int_t)` to set the reference tube (usually 31 for the right array and 37 for the left array), `SetEnergyThreshold(Float_t)` (typically 0.7 to avoid pedestal hits), `SetAdcSel(Float_t)` (usually 0.1, it is the tolerance of the reference hit ADC: $ADC_{ref} = 1 \pm 0.1$), `SetFitWindow(Float_t)` (typically 2ns around the maximum of the peak).

You histogram file should now contain directories `BBx_DeltaTdc` with a summary histogram and a subdirectory with individual tube histograms. See Fig.3 for a typical calibration.

Once this calibration is committed in the database (with the script `CommitDeltaTdc.C`), the slewing correction can be done.

Slewing Correction Calibration

The slewing correction is a refinement of the previous calibration. The purpose is remove the energy (or ADC) dependence of the time signal. This dependence is low for large ADC but can be very pronounced for energy values around the MIP energy, thereofore this will worsen the time resolution. The slewing correction procedure is to estimate the function that describes this time-energy correlation the best and correct the experimental

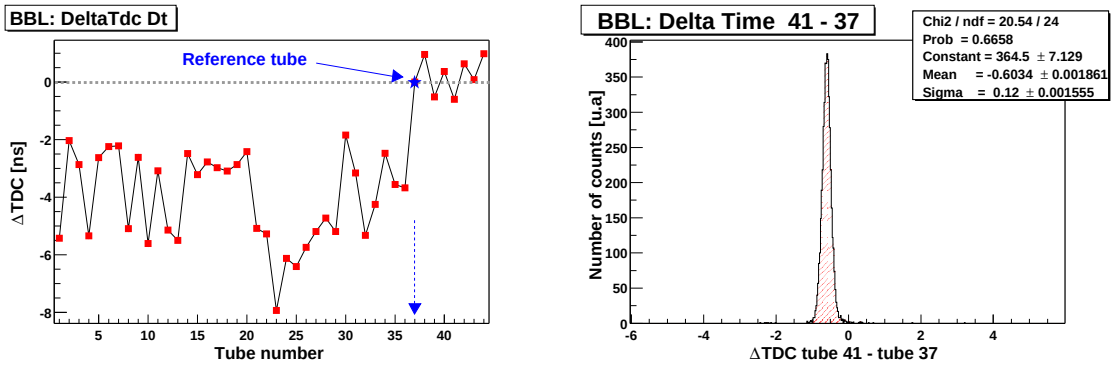


Figure 3: ΔTDC calibration.

signal by removing this dependence. The so far best function we use is

$$\Delta t = t_i - t_{ref} = slewDt + \frac{slewK}{\sqrt{ADC}} + \frac{slewP}{ADC} \tag{1}$$

$slewDt$, $slewK$ and $slewP$ are the slewing calibration parameters. The behaviour of this function is the one described above (the asymptotic limit is slewDt for infinite ADC and is in fact the real ΔTDC).

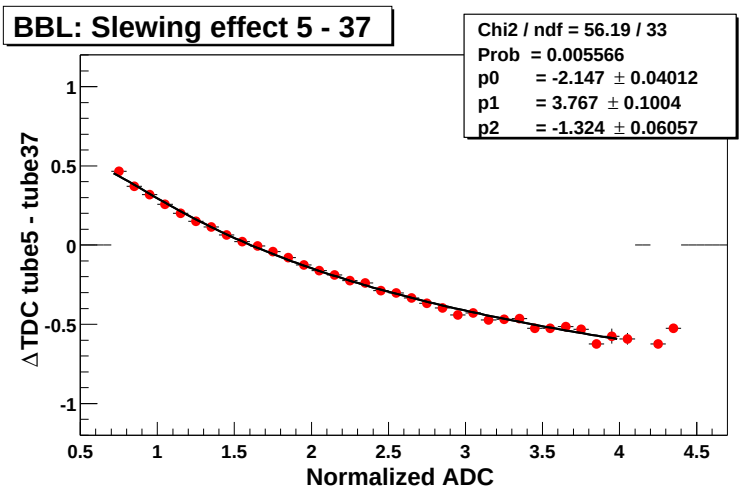


Figure 4: Slewing effect and fitting.

The procedure implemented in brat follows almost exactly the same scheme as the ΔTDC calibration. Use the scripts `BbSlewing.C` and `CommitBbSlewing.C`. The calibration module builds the time difference between the TDC of a given tube and a reference tube but plots it as function of the ADC of this given tube. In principle, for large ADC signals, the correlation curve should get closer to an asymptotic limit, which will be the real time offset. In practice, you might notice some weird correlation for a couple of tubes at these large ADC values (gain saturation ?). If they're really too weird (the function above cannot describe them), you can decide either to set the slewing parameters to 0 (only ΔTDC will be used) or -1111 (the tube will be discarded).

There's a couple of methods that allow you to fixe a limit to your fit range:

```
SetMaxBigTEnergy(Float_t)
SetMaxSmallTEnergy(Float_t)
```

(typically 6 and 3 respectively). You should end up with histograms looking like the one in Fig.4. To quickly check them, cf. Appendix D

Vertex offset

Now that all calibrations are stored in the DB and available, you can proceed with a vertex offset calibration. This offset exists because the time calibration was not absolute. There still remains an unknown delay between the right and left arrays. In order to do it properly, you need TPM1 tracks and raw BB digits (e.g. use the reduced data mentioned above to have noth). The scripts `BbVtxOffset.C` and `CommitVtxOffset.C` do the job. Three offsets are evaluated:

- BB vertex built with the big tubes only
- BB vertex built with the small tubes only
- BB vertex built with the fastest tubes only

Run the script `BbVtxOffset.C` on a tenth of sequences. You will end up with a simple ascii file with offsets and widths. Check the histograms and refit them by hand if needed (correct the values in the ascii files if so). You will note that the small tube vertex is the best correlated (for central events, the width is around 0.6cm) whereas the fastest tube vertex is poorly correlated with TPM1 tracks. On Fig.5, you can see the three correlations. These plots are typical. If you get something totally different, I suspect the previous calibrations were not correct. Commit the values into the DB (you should use the script `CommitVtxOffset.C` and you're done with the BB calibration :)

2.2.2 Track matching offsets

Now comes a very important step. Depending on background conditions, drift velocity fluctuations, etc, the track matching conditions can change, sometimes abruptly. The track matching parameters are:

- `dAly`: difference between the slopes of the incoming and outgoing local tracks in the y direction
- `dAng` difference between the angles of the incoming and outgoing local tracks (related to the bending angle and momentum)
- `dY` difference between the intersections of the incoming and outgoing local tracks on the matching plane (inside the magnet) in the Y direction.

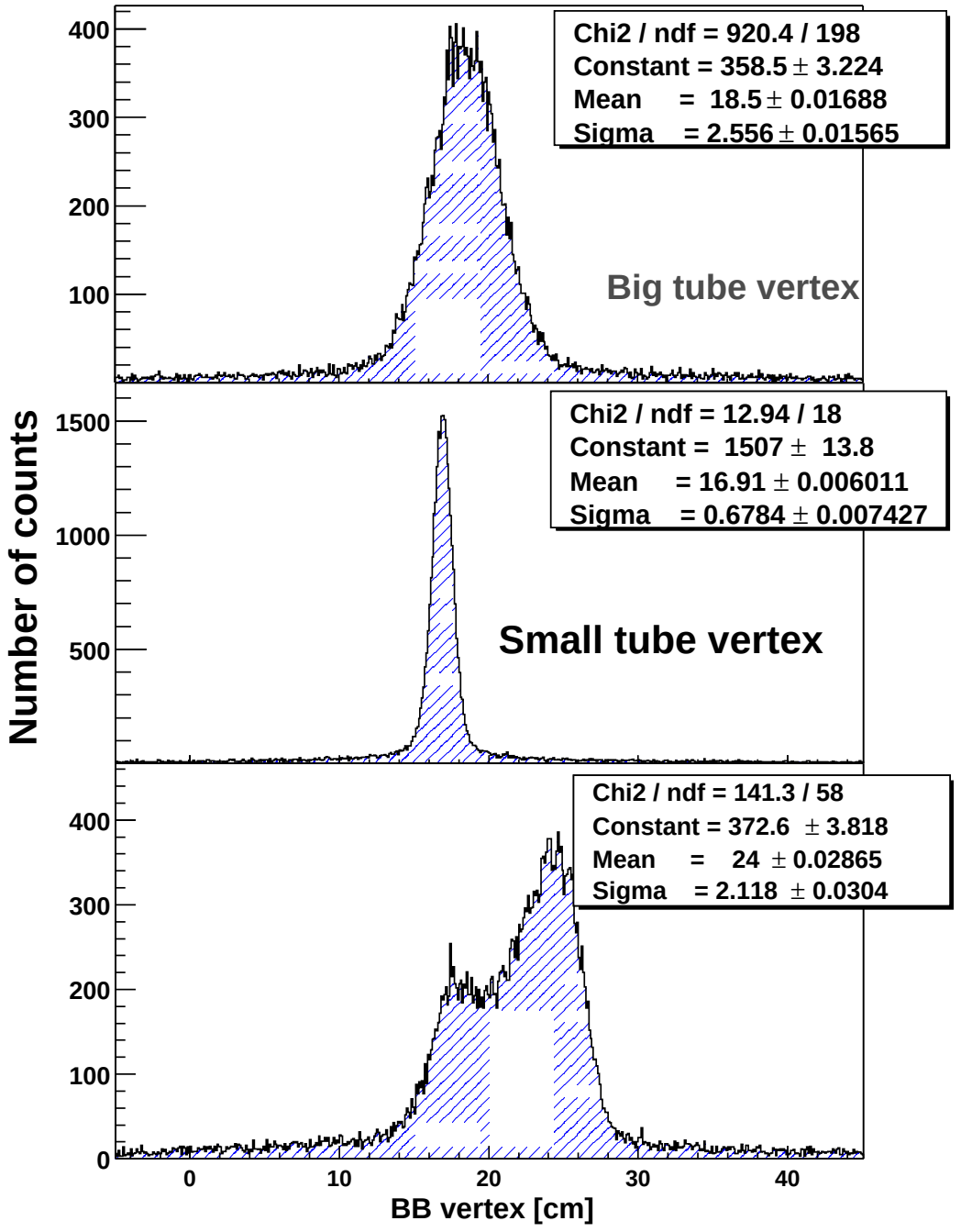


Figure 5: BB vertex offsets for big tubes (top), small tubes (middle) and fastest tubes (bottom).

In principle, if everything was perfect, these parameters should not show any offsets. But of course, perfection doesn't exist in this world :) So we need to know what these offsets are and what the parameter resolution is. There are for now 4 places where we need this information: the MRS, the FFS and two places in the BFS. I have a script called `MatchingOffset.C`. It can be found in `brahms_app/do_app/track`. I usually run this script so that for each run of reduced data, I get a histogram file called `h<run>prel.root`. Run this script with the option `--help` to get some help. In this script, I declare global tracking modules with all offsets set to zero, I also set a loose cut for track selection (15σ for each parameter). You need to have the lastest version of brat to run it because of the introduction of the `BrFsTrackingModule`. For the BFS parameters, *don't clean up duplicates* from T2 or T3 tracks matched with T4 tracks. The purpose is to get a good statistics to evaluate the matching parameter offsets.

Once you have a bunch of histogram files `h<run>prel.root`, you can use the scripts `matchOffset.C` for the FFS and MRS, or `bfsOffset.C` for the BFS. For example, in a root session:

```
root [0] .L matchOffset.C
root [1] offset(<begin run>, <end run>, <spec name>
```

with *<spec name>* = `MRS` or `FFS`. If some runs are missing, don't worry, it will try the next one until it reaches *<end run>*. For the BFS, the script `bfsOffset.C` needs T2_T4 or T3_T4 or B (for back) as an argument for *<spec name>*. The result will consist in 2 things: one is a histogram plot with the three matching parameter offsets and widths as a function of run number (cf. Fig.6), the second is an ascii file `offset`*<brun>*`_`*<erun>*`.`*<spec name>* where all the values are stored. Each line corresponds to a run (see Fig.7). You're now ready for the main data reconstruction, understand global tracking and track–tof hit matching.

3 Main Data Reconstruction

There's nothing much different here (in principle) from the previous step where you already had to recontruct global tracks. You need to know here how to use the track matching offset ascii file. Moreover, you have to add modules that do matching between global tracks and tof hits for future PID. The way to read the ascii file is to first get a module called `BrMatchingOffsetModule` located in `brahms_app/do_app/track`. I suggest you get all my brahms_app directory and compile the subdir track. You can then load the library `libTrackUtil.so` in your own configuration scripts (cf. [2]). Let's take a closer look at the example below for the MRS (adding the FS can be done in the same way):

Here I include the module that updates the DB information.

```
//_____________________________________________________________________
// Module: BrDbUpdateModule
BrDbUpdateModule* dbUpdateModule =
  new BrDbUpdateModule("DB", "DB Update");
mainModule->AddModule(dbUpdateModule);
```

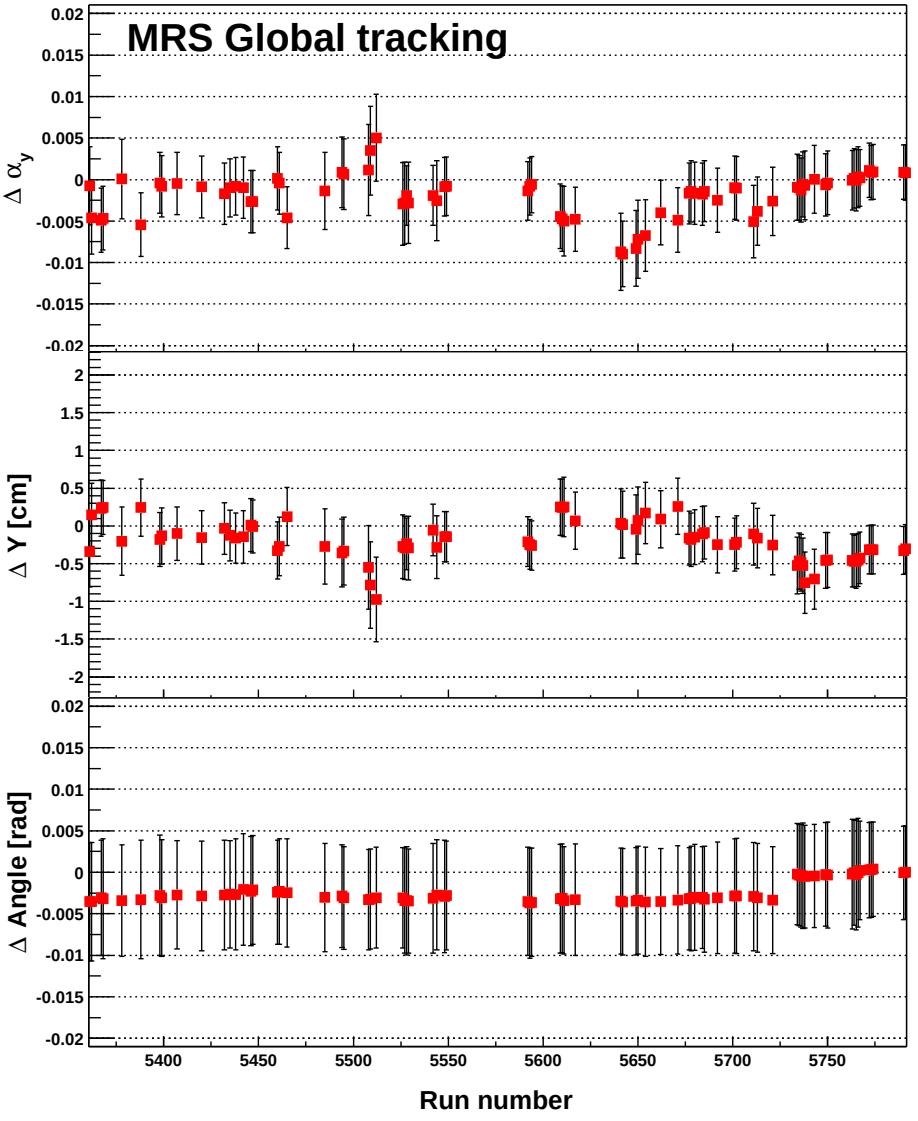


Figure 6: Matching parameters versus run number.

```
*----------------------------------------
* MRS  Track matching offsets and widths
*
*  Run    |   dAlY   Off   -   Sig  |    dY    Off   -   Sig    |  dAng  Off   -   Sig    |
```

Run	dAlY Off	- Sig	dY Off	- Sig	dAng Off	- Sig
5361	−0.0007518	0.0047228	−0.3425198	0.4682708	−0.0035288	0.0071138
5362	−0.0045938	0.0044178	0.1465848	0.4175488	−0.0035318	0.0071358
5367	−0.0049148	0.0038728	0.2343708	0.3731668	−0.0030768	0.0069478
5368	−0.0046488	0.0038268	0.2441048	0.3550178	−0.0031918	0.0072388
5378	0.0000758	0.0047798	−0.2021658	0.4538898	−0.0034148	0.0066918
5388	−0.0054338	0.0038178	0.2440038	0.3792738	−0.0032868	0.0071508
5398	−0.0004028	0.0036728	−0.1784618	0.3574428	−0.0027778	0.0072478
5399	−0.0008308	0.0037018	−0.1331968	0.3720088	−0.0030858	0.0070328
5407	−0.0005018	0.0037518	−0.1016348	0.3544528	−0.0027188	0.0065368
5420	−0.0008928	0.0037018	−0.1545428	0.3547738	−0.0028608	0.0066108
5432	−0.0017128	0.0037018	−0.0338298	0.3427848	−0.0027288	0.0066148
5435	−0.0009988	0.0035128	−0.1273868	0.3338908	−0.0026558	0.0064488
5438	−0.0008088	0.0034828	−0.1622078	0.3336248	−0.0026678	0.0066998
5442	−0.0009858	0.0036878	−0.1460548	0.3451178	−0.0020768	0.0066998
5446	−0.0026658	0.0037338	0.0099908	0.3500168	−0.0022758	0.0065908
5447	−0.0026578	0.0037348	−0.0072848	0.3511558	−0.0021328	0.0065678
5460	0.0001528	0.0037978	−0.3248938	0.3814188	−0.0024068	0.0062608
5461	−0.0004268	0.0037238	−0.2737068	0.3885278	−0.0023018	0.0063568
5465	−0.0046048	0.0037268	0.1241918	0.3839338	−0.0024878	0.0065288
5485	−0.0013718	0.0046578	−0.2731968	0.4987378	−0.0030348	0.0065198
5494	0.0008738	0.0042548	−0.3575988	0.4498748	−0.0028728	0.0061888
5495	0.0006518	0.0042728	−0.3327958	0.4484788	−0.0031018	0.0061778
5508	0.0011298	0.0054798	−0.5500488	0.5544158	−0.0033088	0.0060648
5509	0.0034768	0.0053518	−0.7855648	0.5721798	−0.0032288	0.0060028
5512	0.0050318	0.0052458	−0.9730038	0.5608928	−0.0030498	0.0060718
5526	−0.0029428	0.0050028	−0.2740998	0.4220838	−0.0030678	0.0060608
5527	−0.0028388	0.0049488	−0.2914578	0.4202348	−0.0033878	0.0063688
5528	−0.0019438	0.0035708	−0.2368358	0.3417108	−0.0034358	0.0065388
5529	−0.0028108	0.0048918	−0.2937958	0.4204938	−0.0034558	0.0062838
5542	−0.0019108	0.0036088	−0.0545508	0.3439218	−0.0031368	0.0065938
5544	−0.0025468	0.0048268	−0.2819648	0.4130388	−0.0027288	0.0066228
5548	−0.0008718	0.0035408	−0.1458248	0.3324748	−0.0029138	0.0067818
5549	−0.0008148	0.0035108	−0.1454538	0.3328708	−0.0027898	0.0065688
5592	−0.0013638	0.0035088	−0.2084388	0.3271588	−0.0035078	0.0065238
5593	−0.0008308	0.0034508	−0.2460078	0.3278598	−0.0037148	0.0066408
5594	−0.0006158	0.0034078	−0.2598808	0.3288208	−0.0036208	0.0065348
5609	−0.0044278	0.0038868	0.2480598	0.3719998	−0.0032068	0.0065458
5610	−0.0047048	0.0039328	0.2476548	0.3753118	−0.0031288	0.0065528
5611	−0.0050058	0.0042008	0.2485678	0.3938368	−0.0033938	0.0064608
5617	−0.0047758	0.0038568	0.0687278	0.3781338	−0.0032968	0.0067238
5641	−0.0087108	0.0046658	0.0326128	0.4590478	−0.0034958	0.0063828
5642	−0.0089728	0.0039438	0.0190348	0.4424048	−0.0035568	0.0064228
5649	−0.0082958	0.0045518	−0.0463338	0.4548158	−0.0034968	0.0064778
5650	−0.0071918	0.0047038	0.0700438	0.4464158	−0.0033358	0.0064878
5654	−0.0067228	0.0043118	0.1713588	0.4061638	−0.0035718	0.0065858
5662	−0.0039888	0.0039028	0.0887018	0.3793148	−0.0035218	0.0063678
5671	−0.0048818	0.0038968	0.2592898	0.3709808	−0.0033498	0.0065198
5677	−0.0016638	0.0036618	−0.1609498	0.3546988	−0.0031888	0.0061568
5678	−0.0014148	0.0036838	−0.1814278	0.3534458	−0.0031498	0.0063058
5680	−0.0016468	0.0037558	−0.1505598	0.3628088	−0.0030258	0.0063848
5684	−0.0017698	0.0037448	−0.1084658	0.3675268	−0.0030438	0.0061278
5685	−0.0014038	0.0036688	−0.0878718	0.3492458	−0.0032428	0.0063708
5692	−0.0024838	0.0038488	−0.2500178	0.3705528	−0.0030748	0.0067118
5701	−0.0009828	0.0037898	−0.2488618	0.3508018	−0.0028358	0.0068888
5702	−0.0010598	0.0038408	−0.2167858	0.3530968	−0.0028628	0.0069388
5711	−0.0050488	0.0043718	−0.1089038	0.4088728	−0.0029338	0.0065398
5713	−0.0038128	0.0041058	−0.1615248	0.3956338	−0.0030748	0.0065568
5721	−0.0026088	0.0041138	−0.2533868	0.3933038	−0.0033648	0.0064318
5734	−0.0009248	0.0039948	−0.5250048	0.3737848	−0.0002138	0.0061028
5735	−0.0009978	0.0038758	−0.4646488	0.3733848	−0.0003158	0.0061048
5736	−0.0012548	0.0038488	−0.4866078	0.3773138	−0.0003938	0.0061858
5737	−0.0005758	0.0039288	−0.5254678	0.3661618	−0.0004198	0.0063678
5738	−0.0007078	0.0041438	−0.7533278	0.4049278	−0.0005188	0.0061768
5743	0.0000428	0.0040758	−0.7063118	0.3973818	−0.0004408	0.0061998
5749	−0.0006608	0.0037458	−0.4549068	0.3703498	−0.0002598	0.0062278
5750	−0.0003938	0.0038098	−0.4487708	0.3623358	−0.0003438	0.0063768
5763	−0.0000678	0.0035878	−0.4587198	0.3516168	−0.0002388	0.0066048
5764	0.0001498	0.0035838	−0.4669908	0.3394328	−0.0000468	0.0063388
5765	−0.0000868	0.0036768	−0.4687878	0.3556338	−0.0002968	0.0066238
5766	0.0002848	0.0036758	−0.4504798	0.3492958	−0.0000508	0.0065348
5767	0.0002078	0.0034248	−0.4244518	0.3399888	0.0002348	0.0059178
5772	0.0011058	0.0032858	−0.3166078	0.3191238	0.0002578	0.0057518
5773	0.0008468	0.0032808	−0.3138248	0.3239878	0.0003158	0.0057388
5774	0.0009198	0.0033248	−0.3134498	0.3239438	0.0003878	0.0056898
5790	0.0008708	0.0032758	−0.3247838	0.3180678	−0.0000808	0.0056028
5791	0.0008358	0.0033308	−0.3049388	0.3219708	−0.0000248	0.0056368

Figure 7: Matching parameter offsets for the MRS.

Then comes the trigger filter.

```
//_____________________________________________________________________
// trigger filter
BrTriggerFilter* trig = new BrTriggerFilter("TRIG","Trigger filter");
trig->AddTrigger(1);
trig->AddTrigger(3);
trig->AddTrigger(4);
trig->AddTrigger(5);
trig->AddTrigger(6);
mainModule->AddModule(trig);
```

The header module should also be put here. I need the trigger information for the final DSTs.

```
//_____________________________________________________________________
// header module
mainModule->AddModule(new BrHeaderModule("Header", "Header Module"));
```

This is the way to recontruct the BB vertex. The parameters set here are optimal as far as I could check.

```
// ----- BB rdo module
BrBbCalHitsModule* bbRdo =
  new BrBbCalHitsModule("BB","BB Calibrated hit Module");
bbRdo->SetTreeOn(kFALSE);
bbRdo->SetUseOldCal(kFALSE);
bbRdo->SetMaxTdc(2800);
bbRdo->SetMinTdc(10);
mainModule->AddModule(bbRdo);

// ---- BB vertex module
BrBbVertexModule* bbVtx = new BrBbVertexModule("BB", "BB Rdo Module");
bbVtx->SetTreeOn(kFALSE);
bbVtx->SetMaxTimeDiff(0.5);
bbVtx->SetUseSqlOffset(kTRUE);
mainModule->AddModule(bbVtx);
```

Here, I declare the module that will read the matching offsets. Note also that although the TPC tracking modules return BrTpcTrack objects, the global tracking modules still need BrDetectorTrack. We therefore need to include the CopyDetectorTrack module for all TPCs.

```
//_____________________________________________________________________
// Load my track library
gSystem->Load("libTrackUtil.so");

// ---- copy TPCTrack to DetectorTrack
```

```
Char_t* tpcNames[] = {"TPM1", "TPM2", "T1", "T2"};
for(Int_t i = 0; i < 4; i++)
  mainModule
    ->AddModule(new CopyDetectorTrack(tpcNames[i],
                                      "Copy track module"));

// ---- global tracking offsets
BrMatchingOffsetModule* trkOffset =
  new BrMatchingOffsetModule("Track Offsets", "Track Offsets");
mainModule->AddModule(trkOffset);
trkOffset->SetUseMrs(kTRUE);
trkOffset->SetMrsFile("offset5361_5791.MRS");
```

I create the MRS tracking module but I don't need to specify the offsets anymore. I just have to decide what my cut will be. I have to tell the matching offset module that `mrs` is the pointer of the MRS tracking module.

```
// ----- tracking module
BrMrsTrackingModule* mrs =
  new BrMrsTrackingModule("MRSTrk", "MRS tracking");
mrs->GetCombineModule()->SetFiducialCutDx(1.);
mrs->GetCombineModule()->SetFiducialCutDy(1.);
mrs->GetCombineModule()->SetSigmaCut(3.);

mainModule->AddModule(mrs);
trkOffset->SetMrsModulePtr(mrs);
```

Here is the module that matches TOFW hits and MRS tracks. You can set offsets between TOFW panels and TPM2. There used to be some but with the new geometry there shouldn't be anymore...to be checked.

```
// ----- tof-track matching module
BrMrsTofMatchingModule* mrsMatch =
  new BrMrsTofMatchingModule("MRS", "MRS Track-Tof Matching module");

mrsMatch->SetNtuple(kFALSE);
mrsMatch->SetTdcRange(10, 4000);   // default is [10, 4000]
mrsMatch->SetMaxAdc(100000.);      // default is 10000000

mrsMatch->SetTpm2PanXOffset(0,  0.);    // default is 0
mrsMatch->SetTpm2PanXOffset(1,  0.);    // default is 0
mrsMatch->SetTpm2PanXOffset(2,  0.);   // default is 0
mrsMatch->SetTpm2PanXOffset(3,  0.);    // default is 0
mrsMatch->SetTpm2PanXOffset(4,  0.);   // default is 0
mrsMatch->SetTpm2PanXOffset(5,  0.);   // default is 0
```

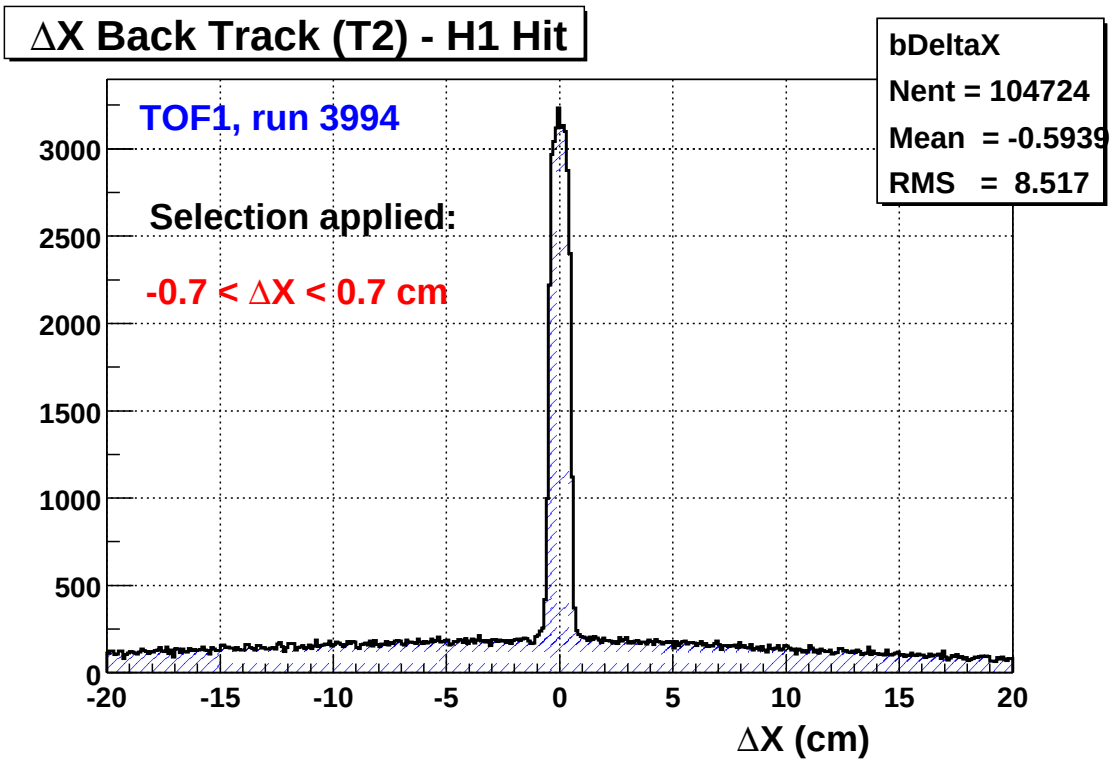


Figure 8: Difference in X between track projections and valid hit positions in the TOF reference frame.

```
mrsMatch->SetMaxDX(1.2);          // default is 0.5
mrsMatch->SetNoPedWidth(10);      // default is 5
mainModule->AddModule(mrsMatch);
```

In Fig.8, there's an example of tof hits matching tracks in the X direction of the tof detector plane. The peak is selected. This selection depends on the method `SetMaxDX(Float_t)`. Don't put a random number, rather, think that this peak should have approximately the width of a slat. For the FS, cf Appendix C.

I need to save TOFW digits. The reason is that I'll use the output data files (with global tracks and raw TOF digits) for TOF calibration. Note that you can add more stuff if you plan to analyze some centrality dependence on whatever, etc. It's up to you to complete it.

```
//_____________________________________________________________
// copy module
BrCopyModule* copy = new BrCopyModule("Copy", "Copy Module");
copy->AddObject("DigTof TOFW");
mainModule->AddModule(copy);
```

4 Hodoscope Calibration

Now that you have some data with BB vertex, global tracks, tof–track match objects, tof digits, Cherenkov PID objects (useful for H1 and H2 calibration), centrality objects if you thought about it too, you're ready to calibrate the hodoscopes. Like for the BB counters, this should be done at least every week of a run period. I say at least because we also want to do it by setting (angle and magnetic field). You should then check with the DAQ run viewer that you don't mix up different run settings.

The TOF calibrations are:

- `TDC Gain`
- `ADC Gain`
- Δ`Delay and Effective speed of light`
- `Time Offset`
- `Slewing Correction`

For each of them, there are 2 scripts in

<brat-install-dir>/share/brat/scripts/tof

One, named `Tofxxx.C`, processes the data and saves teh results into ascii files. The other, named `Commitxxx.C`, allows you to commit these ascii files to the SQL database.

The tof calibration module classes derive also from a base class. Therefore, some methods are common to all of them[1] like `SetSaveAscii`, `SetCommitAscii`, `SetCalibFile`, `SetComment`. You already know what they are useful for.

4.1 TDC Gain

Same story (cf. BB TDC Gain). Use the scripts `TofTdcGain.C` and `CommitTdcGain.C`. Run the 1st one on TDC gain calibration *only*. Do it only once a year (well, do it as many times as there are some TDC gain cal. run to check the gain stability).

4.2 ADC Gain

Same story as well (cf. BB ADC). Use the scripts `TofAdcGain.C` and `CommitAdcGain.C`. Check the histograms before committing anything to the SQL DB. A typical calibrated ADC should look like the one on Fig.9. The only thing to know is that each slat has 2 tubes, one on its top, one at its bottom. You should therefore make twice more checks.

You should be aware that some tubes will show very little statistics (e.g. outer slats in TOFW). If there are only a tenth of counts, you cannot conclude anything. What I did in this case is to check how stable the gains were for tubes with high statistics over a long

[1] In fact, the BB calibration modules were strongly inspired by the TOF modules.

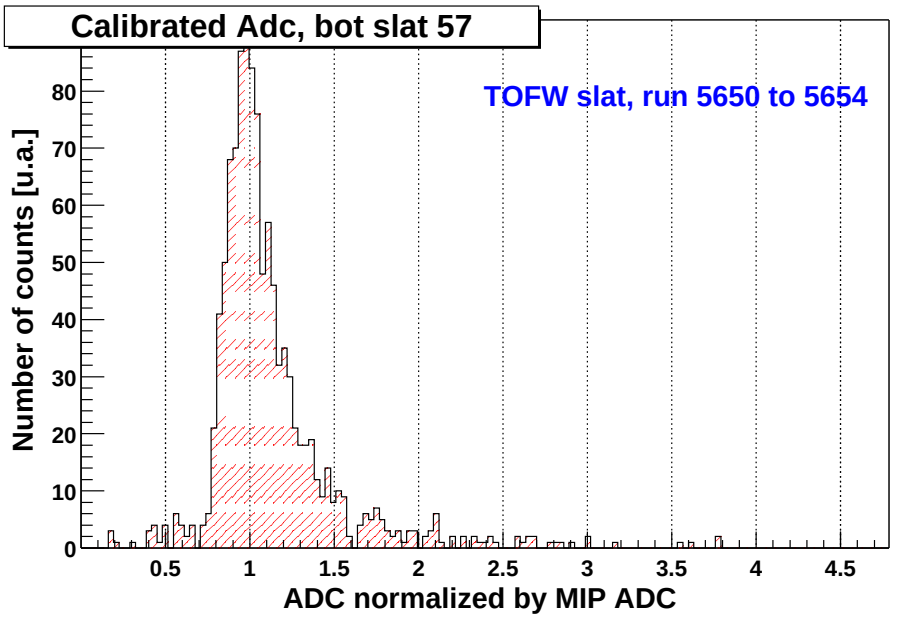


Figure 9: Typical calibrated ADC.

period of time. I then merged all histograms over this period of time (really a lot) to have a reasonable statistics in the outer tubes and could therefore get a rough ADC gain. For this calibration, I think that mixing different settings doesn't really matter.

4.3 ΔDelay and Effective speed of light

This calibration is specific to the hodoscopes. The quantity ΔDelay is the difference between the delays of the bottom and the top tubes. Indeed, each tube has a delay (like the BB tubes). It is possible to get the difference between these delays from simple geometrical considerations. Take tracks matching hits in a given slat. These tracks correspond to particles that deposit energy in this slat. From the intersection between the tracks and the slat, light is emitted and propagates towards the tubes. Since the slat is made of scintillating material and has a certain finite volume, there is some reflections and other processes that contribute to lower the speed of light, that's why we talk about an *effective speed of light*. It turns out that the difference between the time signals of the bottom and top tubes is directly proportional to the location (along the slat axis) of the track intersection. We have the equation

$$\Delta t = t_{bot} - t_{top} = \frac{2Y_{track}}{c_{eff}} \tag{2}$$

with c_{eff} the effective speed of light. But in reality, because of the delays in each tube this equation is

$$\Delta t_{exp} = (t_{bot} - t_{top})_{exp} = \frac{2Y_{track}}{c_{eff}} + \Delta Delay \tag{3}$$

This is illustrated in Fig.10.

So, after having used the script `TofDeltaDelay.C`, you should check all profiles in the root directories `TOFxDelays/Slats` and the summary histograms in `TOFxDelays/Delays`. A

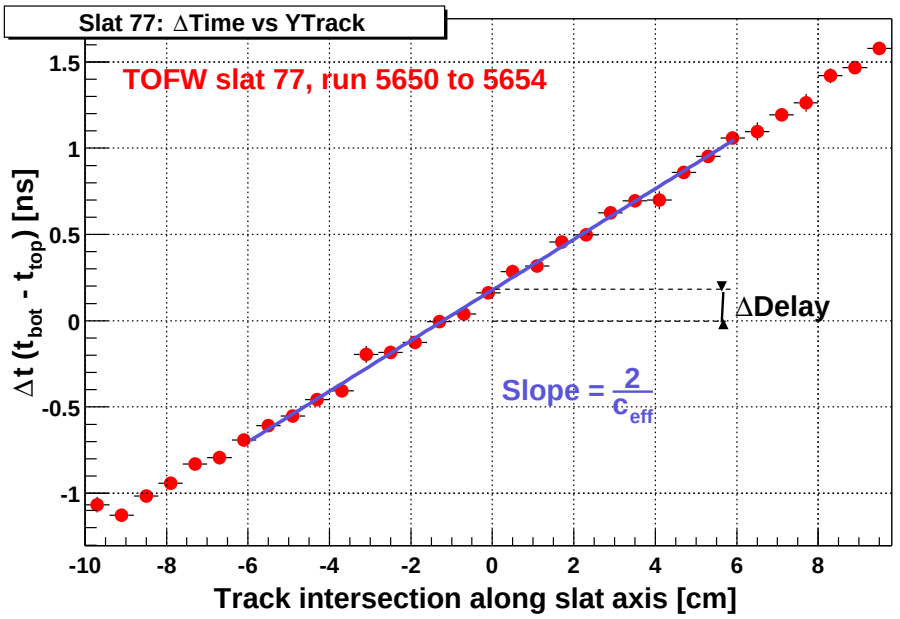


Figure 10: ΔDelay and effective speed of light.

typical summary for H1 is shown on Fig.11. On average, all slats have the same effective speed of light. Yet, a couple of them seem to be lower. This can be due to noisy PMTs, bad fits due to low statistics, etc. You can commit the calibration with `CommitDeltaDelay.C`.

This calibration is useful for selecting tracks well correlated to TOF signals. Indeed, even if the prior selection in the X direction removed most of the background, there still remain some ghost tracks or multiple hits matching single tracks, etc. Selecting tracks and hits matching as well in the Y direction removes a part of this noise. Note that you can set the range of the profile fit with the method

```
BrTofDeltaDelayCalModule::SetYRange(<min>,<max>)
```

<min> and *<max>* are in cm. They shouldn't be bigger than 2/3 of the slat height to avoid fitting non-linearities.

4.4 Time offsets

Like for the beam-beam counters, there's a time offset for each PMT. It is also crucial to correct for this offset in order to get a good TOF resolution and consequently a good PID. But if one manipulates equations relating the time measured by the TDC modules and the real particle time of flight, it is possible then to reduce the PMT time offset to an overall slat time offset (ignoring the slewing correction). Indeed, if b_{off} and t_{off} are the time offsets for the bottom and top tubes respectively, tof the particle time of flight, t_{bot} the time measured by the bottom tube TDC, t_{top} the one measured by the top tube TDC, t_o the starttime of the collision, $b_\perp$ the time it took for the light in the slat to reach the bottom tube and $t_\perp$ to reach the top tube, we have (if we ignore the slewing effect):

$$t_{bot} = tof + b_{off} + b_\perp + t_o \quad (4)$$

$$t_{top} = tof + t_{off} + t_\perp + t_o \quad (5)$$

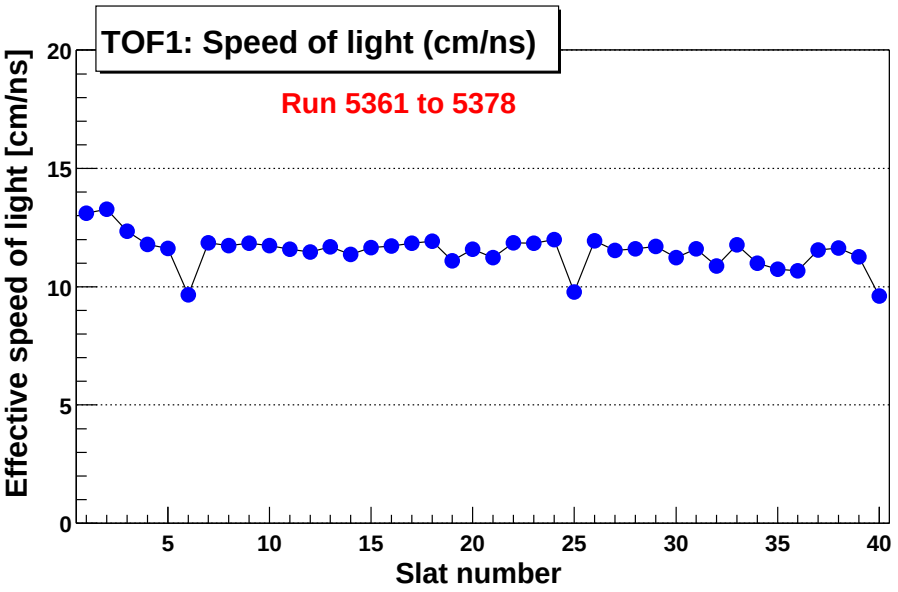


Figure 11: Effective speed of light versus Slat number in H1.

By summing these two equations and isolating tof, one can extract only one constant:

$$tof = \frac{1}{2}\left[t_{top} + t_{bot}\right] - t_o - time_{offset} \tag{6}$$

with $time_{offset} = 0.5(b_{off} + t_{off} + b_{\perp} + t_{\perp})$. Indeed, $b_{\perp} + t_{\perp}$ is a constant since it's equal to $Slat\,Height\,/c_{eff}$. The algorithm implemented in `BrTofTimeOffsetCalModule` is a bit more complicated. Since we don't know either tof or $time_{offset}$, we have to a bit more clever. What we know is that a track is associated to the tof hit. This track has a momentum and a path length. Since most of the particles are pions, we don't introduce (statistically speaking) too much error by assuming that the particle mass is the pion mass. Having this in hands, we can calculate an expected time of flight:

$$tof_{th} = \frac{L_{track}}{c}\sqrt{\frac{p_{exp}^2}{m_{\pi}^2} + 1} \tag{7}$$

It is now trivial to get the time offset:

$$time_{offset} = tof_{exp} - tof_{th} \tag{8}$$

with $tof_{exp} = 0.5\,\left[t_{top} + t_{bot}\right] - t_o$. If you have followed until now, congratulations, you're smart enough to do the TOF calibrations.

Anyway, there are scripts `TofTimeOffset.C` and `CommitTimeOffset.C` to do the job (pick them up at the usual place). For the FS, we have Cherenkov detectors that allow to select only pions. You should therefore perform a Cherenkov PID in your main recontruction (cf. Appendix C). Once the calibration is done, check of course the histograms. A time offset of a given slat should look like the one on Fig.12. The peak is the time offset. The tail is due to particles that are not pions.

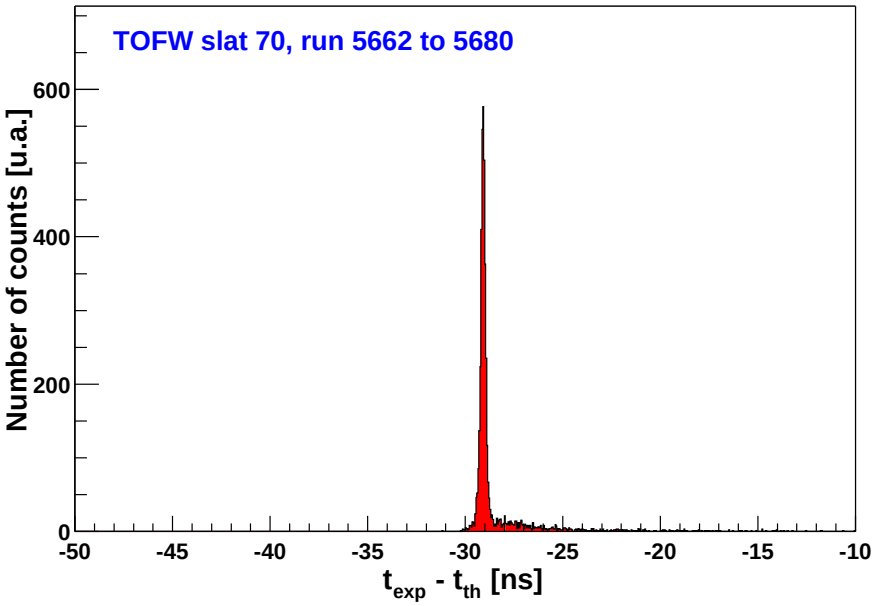


Figure 12: A typical time offset calibration.

4.5 Slewing correction

There's no slewing correction for TOFW. It is not obvious to do if you cannot select a certain particle specie. In the FS, it is possible thanks to the Cherenkov PID. The only problem is that the slewing effect is a property of a tube and cannot really be averaged over the whole slat like for the time offsets. Because of this, you have to rely on your effective speed of light calibration since you need to evaluate the particle time of flight measured by each individual tube. This calibration should for the moment be taken care of by experts only.

5 Final Reconstruction (PID and DSTs)

Now that you have calibrated the hodoscopes, you can reconstruct the tof hits and make a PID. You should pick up the script

`brahms_app/pc_app/dst/makeDst.C`. You have first to compile the libDst (in the same directory). This script has TOF rdo and pid modules for each spectrometer arm. A typical call of this script would be:

```
bratmain makeDst.C -r <run> -I <input-dir> -t <tree-file> -v 5 --fs
```

<input-dir> is the directory where you have your global track sequences, *<tree-file>* is the output tree file (DST), `--fs` enables the job for the forward spectrometer. So far, I never had `--fs --mrs` on the same command line. It's because we agreed originally with Peter Christiansen to have separate trees for MRS and FS. You will have therefore to run the job twice, one for the MRS, one for the FS *separately*.

Once you have DSTs, you can browse them with root scripts in the `brahms_app/pc_app/dst` directory (called `exeAnaxxxDst.C`). The rest is up to your imagination and intelligence

for the analysis. To know more about the DSTs (trees containing all the information for physics analysis), contact me or Peter at `ouerdane@nbi.dk, pchristi@nbi.dk`.

References

[1] Christian Holm, *CRASH, CRs Application SoftwareHome*, `http://pii3.brahms.bnl.gov/~brahmlib/crash/`

[2] Christian Holm, *The Hitchhikers Guide to BRAT*, in `<install-dir>/share/doc/brat`

[3] Yury Blyakhman, *Beam-Beam Counters and Charged Particle Multiplicity in the Brahms Experiment*, Department of Physics, New York University, May 2001

A Database Connection

A typical sequence in the bratmain configuration script:

```
//____________________________________________________________
//
// Database section
//
BrMainDb* mainDb = BrMainDb::Instance();
mainDb->SetUserName(dbuserOption->GetValue());
mainDb->SetHostName(maindbOption->GetValue());
mainDb->SetDbName("BrahmsMain");

// Un comment if you need access to database
if (!mainDb->Connect())
  return;
if (!mainDb->ConnectToRun())
  return;
if (!mainDb->ConnectToCalib())
  return;
if (!mainDb->ConnectToGeom())
  return;

BrRunInfoManager* runInfoManager =
  BrRunInfoManager::Instance();
runInfoManager->SetDebugLevel(debugOption->GetValue());
runInfoManager->Register(runOption->GetValue());

BrCalibrationManager* calibrationManager =
  BrCalibrationManager::Instance();
calibrationManager->SetDebugLevel(debugOption->GetValue());

BrGeometryDbManager *geometryManager =
  BrGeometryDbManager::Instance();
geometryManager->SetDbModeMySQL();
```

If you don't know anything about it, refer to [2] but the default values should be ok if your `$(HOME)/.bratdbrc` file is up to date.

It can happen that pii3 is down. In this case, the connection sequence is:

```
//____________________________________________________________
//
// Database section
//
BrMainDb* mainDb = BrMainDb::Instance();
mainDb->SetUserName(dbuserOption->GetValue());
mainDb->SetHostName(maindbOption->GetValue());
```

```
  mainDb->SetDbName("BrahmsMain");

  BrRunsDb* rundb = BrRunsDb::Instance();
  rundb->SetUserName("query");
  rundb->SetHostName("rcas0005.rcf.bnl.gov");
  rundb->SetDbName("RUNDB");

  // Un comment if you need access to database
  if (!mainDb->Connect())
    return;
  if (!rundb->Connect())
    return;
  if (!mainDb->ConnectToCalib())
    return;
  if (!mainDb->ConnectToGeom())
    return;
```

The difference with the first sequence is that by default the BrMainDb will connect to pii3. In the last case, you explicitely set the host name where a copy of the RUNDB sits.

B Hints on Reconstruction Production

There are many ways to reconstruct many sequences of many runs. For production reconstruction, I prefer doing it sequence by sequence (safer!) since jobs can take long and you don't want them to crash after the 100000th event when you output the result to one single file...which makes it unreadable. Since the output module (`BrEventIO`) has its io mode set to `BrIOModule::kBrJobFile | BrIOModule::kBrRecreateFile`, it can only output to one single file. Therefore, if you want to get output files sequence by sequence, you need e.g. a shell script in which you would make a loop over the number of sequences. An example of such a shell script can be found on rcas in `~ouerdane/cal/reco/Reco.csh`. If you call the help option, you would get:

```
rcas0020:reco> ./Reco.csh -h
 ------------ Command options -----------------------------
 -c  --script       <scr>    Script filename
 -r  --run          <run>    Run number, default is 0
 -s  --sequence     <seq>    First sequence, default is 0
 -f  --final-seq    <fseq>   Final sequnce number, default is 0
 -e  --events       <nevt>   Number of events default is 100000000000
 -i  --input-dir    <dir>    input directory, default is data1
 -o  --output-dir   <dir>    output directory, default is reco2/ffs
 -g  --geo-basename <geo>    Geofiles basename (<base>.geo <base.mag>)
 -h  --help                  This help
 ---------------------------------------------------------

 ... Ciao!!
rcas0020:reco>
```

A typical call of this script would then be:

```
./Reco.csh -c <bratmain-script> -r <run> -s <begin-seq> -f <final-seq>
-i <input-dir> -o <output-dir>
```

For each sequence in *<input-dir>*, you would get a output sequence in *<output-dir>*. I do this systematically for the main reconstruction (cf. section 3).

C Reconstruction of the FS data for TOF calibration

In section 3, I gave an example dealing with the MRS. For the FS, it is quite similar in essence, but you have to deal with more things. Let me show you how it is done:

First, I load my track library and set the approriate spectrometer sections:

```
//____________________________________________________________________
// Load my track library
gSystem->Load("libTrackUtil.so");

// ---- copy TPCTrack to DetectorTrack
Char_t* tpcNames[] = {"TPM1", "TPM2", "T1", "T2"};
for(Int_t i = 0; i < 4; i++)
  mainModule
    ->AddModule(new CopyDetectorTrack(tpcNames[i],
                                      "Copy track module"));

// ---- global tracking offsets
BrMatchingOffsetModule* trkOffset =
  new BrMatchingOffsetModule("Track Offsets", "Track Offsets");
mainModule->AddModule(trkOffset);
trkOffset->SetUseFfs(kTRUE);
trkOffset->SetUseBfs(kTRUE);
trkOffset->SetFfsFile("<some offset file for ffs>");
trkOffset->SetBfsFiles("<t2-t4 file>", "<t3-t4 file>", "<t4-t5 file>");
```

I then declare a `BrFsTrackingModule` and call `UseBbVertex()` so that it gets the BB primary vertex for tracking back to it and I don't forget to tell `trkOffset` which are the pointers to `BrFfsTrackingModule` and `BrBfsTrackingModule`. This is important if you want to use the track matching offsets ascii files.

```
// FS Tracking module
BrFsTrackingModule* fsmod =
  new BrFsTrackingModule("FS", "FS Tracking");
fsmod->UseBbVertex();
mainModule->AddModule(fsmod);

// FFS
```

```
  BrFfsTrackingModule* ffs = fsmod->GetFfsModule();
  trkOffset->SetFfsModulePtr(ffs);
  ffs->GetCombineModule()->SetFiducialCutDx(1.);
  ffs->GetCombineModule()->SetFiducialCutDy(1.);
  ffs->GetCombineModule()->SetSigmaCut(3.);
  ffs->SetNtuple(kFALSE);

  // BFS
  BrBfsTrackingModule* bfs = fsmod->GetBfsModule();
  trkOffset->SetBfsModulePtr(bfs);
  bfs->SetExtendD3Match(kFALSE);
  bfs->SetExtendD4Match(kFALSE);
  bfs->SetCleanUp(BrBfsTrackingModule::kCUT2, kTRUE);
  bfs->SetCleanUp(BrBfsTrackingModule::kCUT3, kFALSE);
  bfs->SetD4FieldScaleFactor(1.0);

  // FRONT
  bfs->GetCombineT2T4()->SetFiducialCutDx(1.);
  bfs->GetCombineT2T4()->SetFiducialCutDy(1.);
  bfs->GetCombineT2T4()->SetSigmaCut(3.);

  bfs->GetCombineT3T4()->SetFiducialCutDx(1.);
  bfs->GetCombineT3T4()->SetFiducialCutDy(1.);
  bfs->GetCombineT3T4()->SetSigmaCut(3.);

  // BACK
  bfs->GetCombineT4T5()->SetFiducialCutDx(1.);
  bfs->GetCombineT4T5()->SetFiducialCutDy(1.);
  bfs->GetCombineT4T5()->SetSigmaCut(3.);
```

That should be it for the global tracking modules. The rest deals with tof–track matching. Here we go:

```
 // ----- tof-track matching module
  BrFfsTofMatchingModule* match =
    new BrFfsTofMatchingModule("FFS", "FFS Track Matching module");

  match->SetUseH2(kFALSE);
  match->SetNtuple(kFALSE);
  match->SetTdcRange(10, 4000);   // default is [10, 4000]
  match->SetMaxAdc(100000.);      // default is 0.5
  match->SetT2H1XOffset(0.);      // default is 0
  match->SetMaxH1DX(0.7);         // default is 0.5
  match->SetNoPedWidth(10);       // default is 50
  mainModule->AddModule(match);

  BrBfsTofMatchingModule* bmatch =
    new BrBfsTofMatchingModule("BFS", "BFS Track Tof Matching module");
```

```
bmatch->SetNtuple(kFALSE);
bmatch->SetTdcRange(10, 4000);   // default is [10, 4000]
bmatch->SetMaxAdc(100000.);      // default is 0.5
bmatch->SetMaxH1DX(0.7);         // default is 0.5
bmatch->SetMaxH2DX(1.);          // default is 0.5
bmatch->SetT2H1XOffset(0.);      // default is 0
bmatch->SetT3H1XOffset(0.);      // default is 0
bmatch->SetT5H2XOffset(0);       // default is 0
bmatch->SetNoPedWidth(10);       // default is 50
mainModule->AddModule(bmatch);
```

For time offsets calibration and slewing correction, you should have a Cherenkov PID as well. In your script you should have *after* the global tracking modules. You should contact Claus Ekman at `ekman@nbi.dk` for the latest and best set of parameters. Don't take the ones below as granted.

```
// C1 RDO
BrChkvRdoModule* c1Rdo =
  new BrChkvRdoModule("C1","C1");
c1Rdo->SetThreshold(1.0);
mainModule->AddModule(c1Rdo);

// C1 PID
BrC1PidModule* c1PidMod =
  new BrC1PidModule("C1Pid","C1Pid");
mainModule->AddModule(c1PidMod);

// RICH RDO
BrChkvRdoModule* richRdo =
  new BrChkvRdoModule("RICH","RICH");
mainModule->AddModule(richRdo);

// RICH PID
BrRichPidModule* richPidMod =
  new  BrRichPidModule("RichPid","RichPid");
mainModule->AddModule(richPidMod);
```

Of course, you should not forget to copy tof digits!

```
BrCopyModule* copy = new BrCopyModule("Copy", "Copy module");
copy->AddObject("DigTof TOF1");
copy->AddObject("DigTof TOF2");
mainModule->AddModule(copy);
```

D Calibration check

There's an easy way to know if a revision already exists in the calibration database. You just have to open your favourite web browser, go to this link:

<http://pii3.brahms.bnl.gov/~daq/calibtool>

enter the run number to check and choose the detector(s) you're working on, e.g. in Fig. 13 for run 5662 and BB Left:

Parameter	Start run	End run
pedestal	5789	5789
pedestalWidth	5789	5789
adcGain0	5806	5806
adcGain1	−1	−1
adcGain2	−1	−1
adcGapStart	2484	2484
adcGap	2484	2484
tdcGain	3800	3935
deltaTdc	5806	5806
slewK	5806	5806
slewDt	5806	5806
slewP	5806	5806
timeOffset	−1	−1

Figure 13: Revision check of Beam-Beam Left for run 5662.

The red parameters are the ones without any revision. The 1st column with run numbers is the start time of the revision validity for the corresponding parameter, the last column is the end time of the validity. Note that the policy for most of the calibration parameters is that if a revision is not found in the database, the closest one backward in time to the run you're processing will be used.

A few words on how to check a calibration (histograms), and redo a fit for complicated calibrations. In most of the calibrations, the fit uses a simple gaussian or 1st degree polynomial functions. This can be redone by hand. For more complicated calibrations, I provide a simple root script that allows you to display the histograms + the fits and to refit them. So far, I installed a script called `refitSlewing.C`. Let's say that you have run the script `BbSlewing.C` on data and got a file `bbSlewing.root` containing all the

histograms. The following sequence shows you an example of fit checking:

```
prompt% root bbSlewing.root
root[0]
Attaching file bbSlewing.root...
root[1] .L refitSlewing.C
root[2] display(1, ``l'')          // display tube 1 of left array
root[3] refit(0.7, 3.5, 1, ``l'') // ADC fit range in MIP units
root[4] display(15, ``r'')         // display tube 15 of right array
root[5] refit(0.8, 2.8, 15, ``r'')
root[6] .q
Leaving ROOT
```

Each time you redo a fit if necessary (when e.g. the fit at large ADC is bad), you have to save the new parameters in the ascii file you got when your job was finished. Then you can commit the calibration to the database.

I will provide such a script for other calibrations that require a refitting.