Vertex-determination from TPM1

Bjørn H. Samset

September 8, 2000

A note describing the current status of the vertex-reconstruction from data from TPM1, and how to use the vertexfinder-classes.

Contents

1 Brief outline and conclusion

Section 2 gives a crash course in how to use the classes needed for vertex-reconstruction with TPM1. Those not interested in the technical details can skip this chapter.

Sections 3 and 4 describe the two algorithms in a bit more detail, and I show some plots of the efficiency and limitations of using this detector for vertex-finding. Section 5 then shows the correlations I find with the beam-beam counters (BB's) and the zero-degree calorimeters (ZDC's), and finally section 6 sums up some issues and conclusions.

For the impatient, the main conclusion of this note is that both vertexfinder-modules work well and give nice correlations both internally and with the global detectors. The cluster-module currently has the best efficiency, but this may change if we can improve tracking in TPM1.

Please note that all the analysis presentet here is done with MRS at 90 degrees. Testing of other angle-settings will be done over the next few weeks.

2 Using the classes

2.1 C++ syntax

To reconstruct the vertex from TPM1, do the following:

First, create the vertexfinder-module by either

BrTPMClusterVertexModule *c_vtx
= new BrTPMClusterVertexModule("VTX","VertexFinder","TPM1");

or

BrTPMTrackVertexModule *t_vtx
= new BrTPMTrackVertexModule("VTX","VertexFinder","TPM1");

(See below for details about the two algorithms.)

You also need a clusterfinder set up for TPM1, so do this:

BrTPCClusterFinder *clusterfinder = new BrTPCClusterFinder(“TPM1”,"ClusterFinder");

clusterfinder->SetMaxPixelCutoff(20.0);

The last command is related to the response of the TPC, and will probably change as we get to know it better. Ask an expert for the current value of this and other settable parameters.

2.1.1 BrTPMClusterVertexModule

If you use this module, you are now ready to do reconstruction. In your event-loop, do the following:

clusterfinder->Event(event); to create clusters in the form of BrDetectorHits from the raw TPC-data. Here, 'event' is a pointer to a BrEvent-object filled by BrRawDataInput.

c_vtx->SetDetectorHits(clusterfinder->GetDetectorHits()); to pass the clusters to the vertex-finder.

BrEventNode *reco_node = new BrEventNode("Reconstruction Node");

cluster_vertexfinder->Event(0, reco_node); Create an event-node to store the vertex-information in, and run the vertexfinder. Any BrEventNode will do here - you could even pass the event itself, or any other node you use for reconstruction-information.

The vertex has now been found, and information is placed in the BrEventNode in the form of a BrVertex-object called "VtxData". To get hold of it, do

*BrVertex *vdata = (BrVertex*)reco_node->GetObject("VtxData");*

Float_t vtx_y = vdata->GetY();

Float_t vtx_z = vdata->GetZ(); Get the Y and Z location in cm.

Float_t vtx_y_rms = sqrt(vdata->GetVarianceY());

Float_t vtx_z_rms = sqrt(vdata->GetVarianceZ()); Get an error-estimate in cm.

If you wish to visually inspect the histograms that determine the coordinates, do

c_vtx->SetDrawHistogram(1);

before calling the Event()-procedure. Another command that may be useful is

c_vtx->SetVertexMinNoPoints(n);

which allows you to set the minimum number of clusters in the TPC. If the number is less than n, the algorithm gives up without trying. If this happens, z is set to 999, so ALL ANALYSIS WITH THIS MODULE SHOULD BE MADE WITH THE CONDITION Z<999.

2.1.2 BrTPMTrackModule

If you instead wish to use the track-based vertexfinder, you also need a trackfinder for TPM1. Before the event-loop, do

*BrTrackStringFinder *trackfinder = new BrTrackStringFinder("TPM1","TPM1TrackFinder");*

There is also another trackfinder, BrTrackFollowFinder, which is quicker but currently does not work for TPM1 due to the amount of noise. You could also use Peter Christiansen's BrTrackFollowFinderModule with the 3rd parameter in the creator set to kTRUE.

Whichever tracking-method you use, do the following in the event-loop:

clusterfinder->Event(event); to create clusters from the raw TPC-data.

trackfinder->SetDetectorHits(clusterfinder->GetDetectorHits()); to pass the clusters to the trackfinder.

*BrEventNode *reco_node = new BrEventNode("Reconstruction Node");*

trackfinder->Event(0, reco_node); Find tracks, put them in reco_node.

t_vtx->SetDetectorTracks(trackfinder->GetDetectorTracks()); Pass the tracks to the vertexfinder

t_vtx->Event(0,reco_node); Do the vertexfinding!

*BrVertex *vdata = (BrVertex*)reco_node->GetObject("VtxData");* As above, get a BrVertex-object with all information.

The y- and z-coordinates are extracted as above, but in this case the uncertainty (a χ^2) is found in this way:

z_chisq = vdata2->GetVertexChisq();

Parameters to play with here are

t_vtx->SetVertex_dist_cut(f);

t_vtx->SetVertex_chisq_cut(f);

which control the way the algorithm cuts away tracks. The default values are 2 cm for both cuts - this is probably too high, but I have not been able to play enough with this one to suggest new parameters.

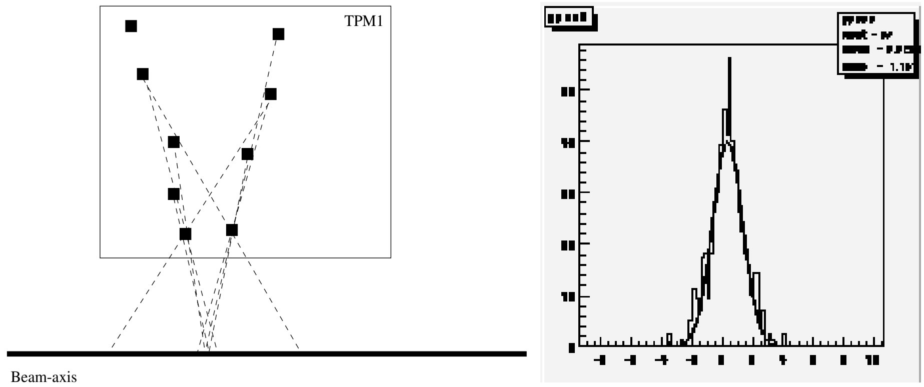

Figure 1: Lines drawn between clusters intersect the beam-axis (left) and give a histogram that can be fitted (right).

2.2 VertexNtuple

If you want to see the classes in action, please refer to the program VertexNtuple which will soon be put in CVS. The file VertexNtupleTPC.cxx contains a tidy example of the syntax, and the accompanying VertexNtupleWriteup.ps gives a few more details.

2.3 BrVertex

In the final analysis we should probably generate a BrVertex-object from the BB's and ZDC's also, to keep the information on the same format. This is not implemented yet, but will not be difficult to add once we agree on how to represent the error-estimate from these detectors.

PLEASE NOTE THAT THIS WILL INCLUDE RENAMING THE BrVertex-OBJECTS GENERATED BY THE ALGORITHMS. SINCE THIS WILL BREAK OLDER VERTEX-CODE I WILL GIVE A LOUD WARNING BEFORE WE DO THIS.

3 BrTPMClusterVertexModule

3.1 How does it work

BrTPMClusterVertexModule works only with clusters from TPM1, i.e. without first doing tracking. It draws a line between clusters in the back and front of the TPC, extrapolates this line to the beam-axis and increment histograms of where it intersects the x=0 plane. A Gaussian fit to these histograms then gives the y- and z-location of the vertex - see ill. in figure 1.

To reduce both CPU-time and background, the TPC is divided up into a settable number of slices in a fan outward from y=0, and to get z I only combine hits within these slices. This can be done because we know that our tracks are straight lines, and so will go from the height y and through such a slice. That we can assume y=0 has been checked by running the algorithm without slicing.

Finally, I also remove a linear background from the histograms to give a clear peak. This may cause a slight problem - see section 6.

This method does not give a value for x, as the angles between the lines become too acute.

3.2 Error-estimate

The error-estimate from BrTPMClusterVertexModule is the RMS of the determining histogram. In events where the algorithm is unable to find a good vertex (as seen by visual inspection of the histogram) this value is at least > 1cm. I use

$$RMS < 0.5cm$$

as a criterion for a "good" vtx-determination - this value will have to be discussed, because I can see by visually inspecting the histograms that this discards a number of well-determined vertices. Once we get PID, we should be able to tune this cut by seeing how high we can go before the mass-peaks smear out.

3.3 Performance

As can be seen in figure 2, the algorithm is mostly able to fit nice Gaussians in both y and z. I have also included one bad fit, to show how this makes a large RMS.

In figure 3, I have analyzed 10279 events (seq 000) of run 2235 which has MRS at 90 degrees. There are 1925 trigger 6-events in the file, and I have preselected on events where TPM1 has more than 50 clusters so that the vertexfinder can work.

The first two histograms show the y- and z-distributions of the vertex. Y is nicely centred on 0. The z-distribution, which is shown in a log-scale, shows all reconstructed vertices in a light shade and all vertices satisfying RMS$<$0.5 in a dark shade. The ratio between these for all events is roughly $\frac{1}{10}$ (see figure 3, picture 2), and the absolute ratio of reconstructed vertices to trigger 6-events is $\frac{445}{1925} = 0.231 \approx 23\%$.

The z-distribution shows the acceptance of ± 40 cm that is introduced at one point in the algorithm. This is done to save CPU-time and only reconstruct a narrow fedducial area - I will remove this cut for the real analysis, and this will probably also heighten the efficiency.

The next four histograms show the distribution of RMS-values in y and z for all events (2nd row) and trigger 6-events (3rd row). The y-determination seems good to within 0.5 cm in almost all events, while the z-determination is not as good. These pictures should help us decide what value to use as an RMS-cut.

4 BrTPMTrackVertexModule

4.1 How does it work

BrTPMTrackVertexModule requires at least two tracks in TPM1. It projects these tracks back to the beam-axis (where they are assumed to be closest), and

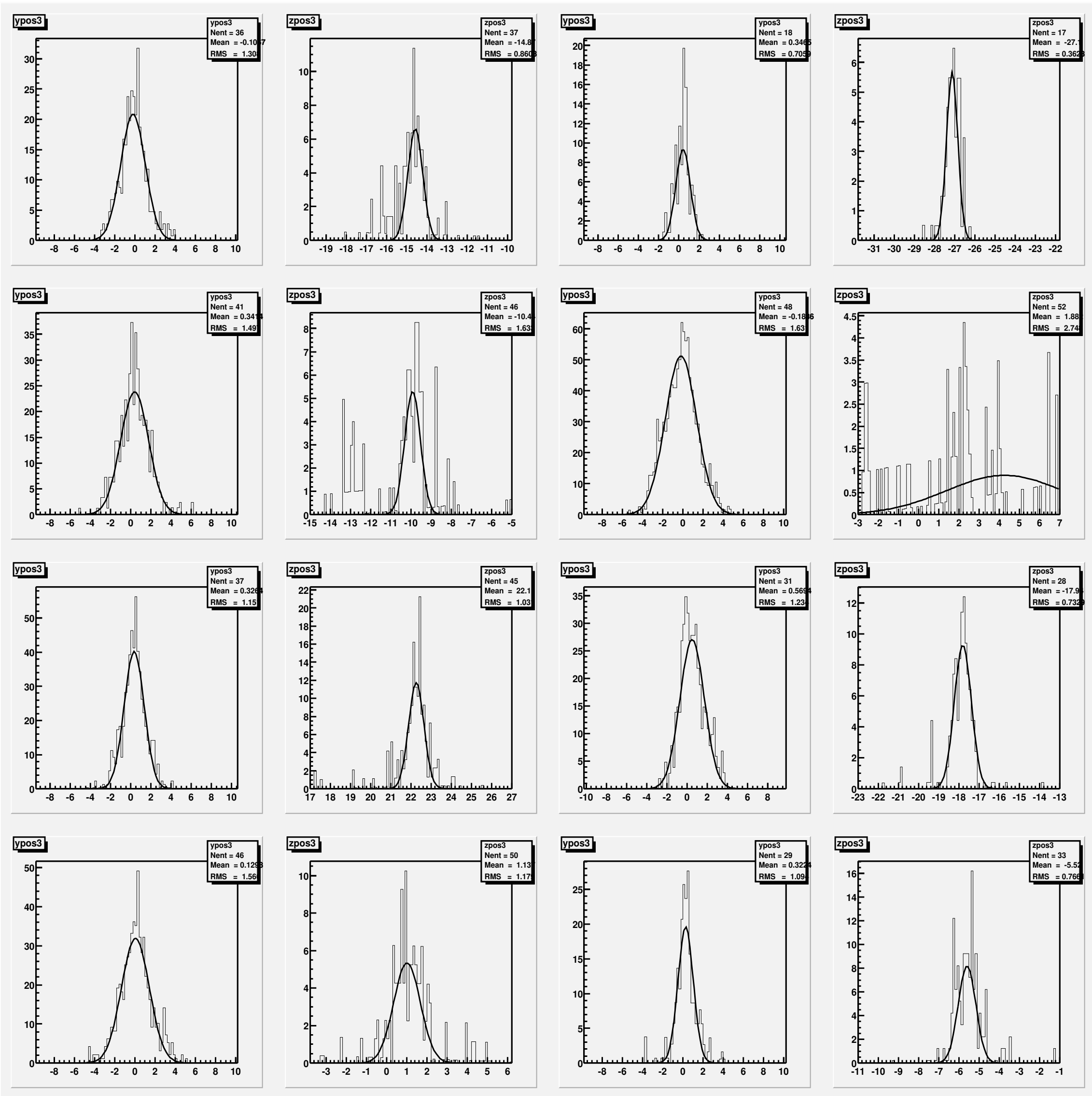

Figure 2: Histograms determining y (col's 1 and 3) and z (col's 2 and 4). Also note the one "bad" z-fit, resulting in a large RMS-value. (Data from run 1589, i.e. before the open period.)

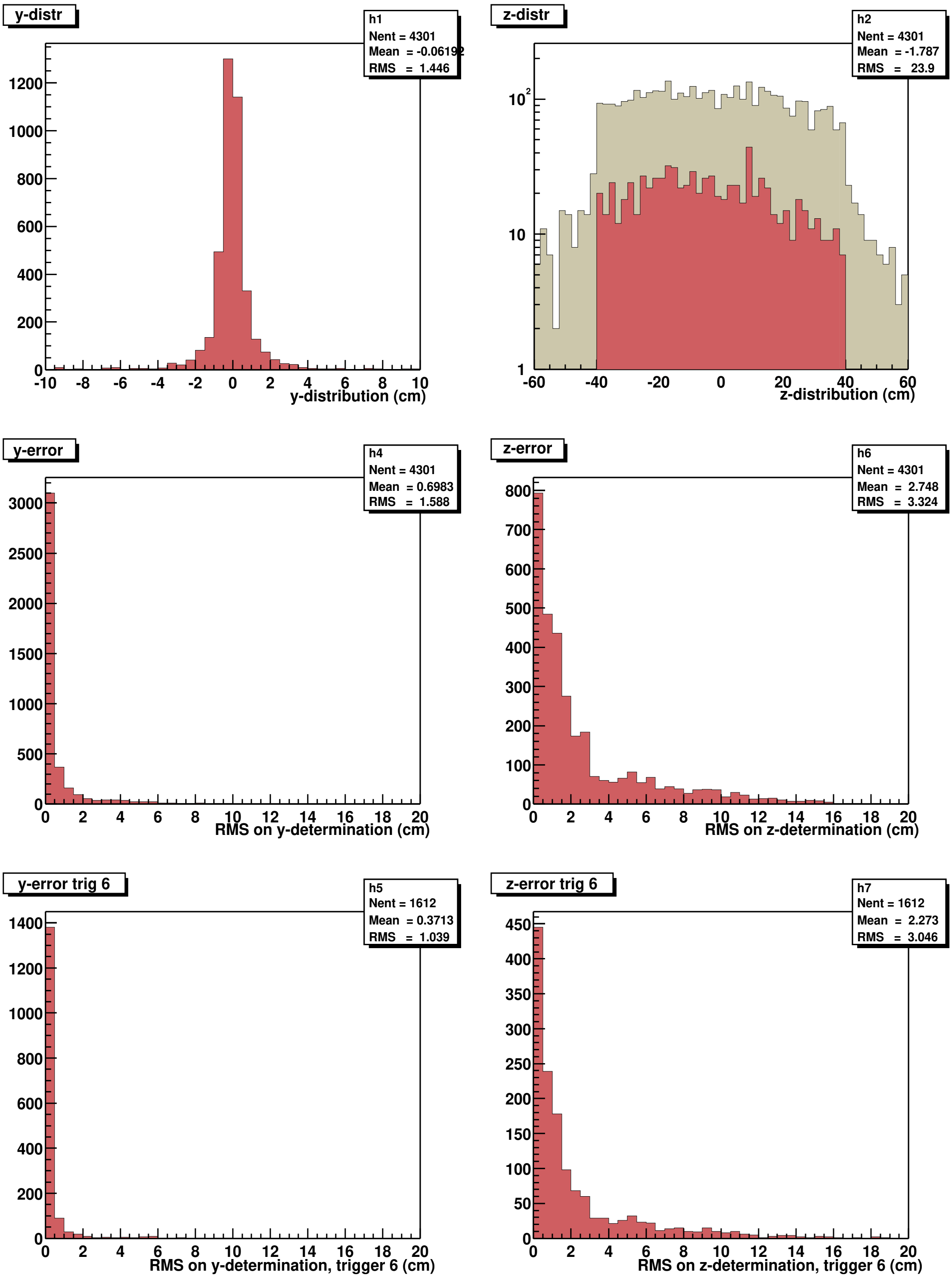

Figure 3: BrTPMClusterVertexModule-results from run 2235, seq 0.

via a least-squares algorithm finds the 3-dimensional point that minimizes the distance to all tracks. Then it enters a loop where it removes tracks outside a settable cut-distance from this minimized point, runs the least-squares algorithm again to see if the χ^2decreases, and if it does tries to remove more tracks etc. This gives a good determination of the vertex in y and z, but because of the small angles between the tracks this method is also unable to determine x (the uncertainties in the track-fitting completely dominate).

4.2 Error-estimate

The error-estimate is a χ^2 calculated from the shortest distance from the tracks to the determined vertex-point. I have not tested this as extensively as for the other module due to problems with tracking in TPM1, but performance-tests (see below) hint at

$$\chi^2 < 5$$

being a good criterion for a “well-determined” vertex.

4.3 Performance

Figure 4 shows the reconstruction-results for the same run as in section 3.3, i.e. 10279 events with MRS at 90 degrees.

In the y- and z-distributions the light-shaded histogram is for all events with two or more tracks, while the dark-shaded histogram has the constraint $\chi^2 < 5$. It seems that the efficiency may be declining towards the edges of the histogram, but it is difficult to tell because of the many high-χ^2events around z=0. This point requires some more testing.

The ratio of reconstructed vertices to number of events for trigger 6 is here $\frac{230}{1925} = 0.119 \approx 12\%$. I expect this will increase as we improve tracking in TPM1 and tune the parameters of the algorithm a bit more.

The two lower pictures are the reason why I chose $\chi^2 < 5$ as a condition. It may be that some other cut is better - please let me know if you have any thoughts on this.

5 Correlations with BB and ZDC

Since we have three detectors capable of producing vertex-information, it is essential that these three agree. Below I show the correlations I find when analyzing the 10279 events in run 2235 seq 000. All values from TPM1 are under the constraint that RMS < 0.5 for the clustermodule and $\chi^2 < 5$ for the trackmodule.

5.1 BB vs ZDC

5.1.1 Correlations

Just to show that the BB's and ZDC's are consistent, figure 5 shows the correlation between them. In this analysis, the ZDC's seem to be shifted by roughly 30 cm (1 ns) relative to the BB's - I have not found the reason for this, but I see the same thing with the TPC-data (see 5.3)

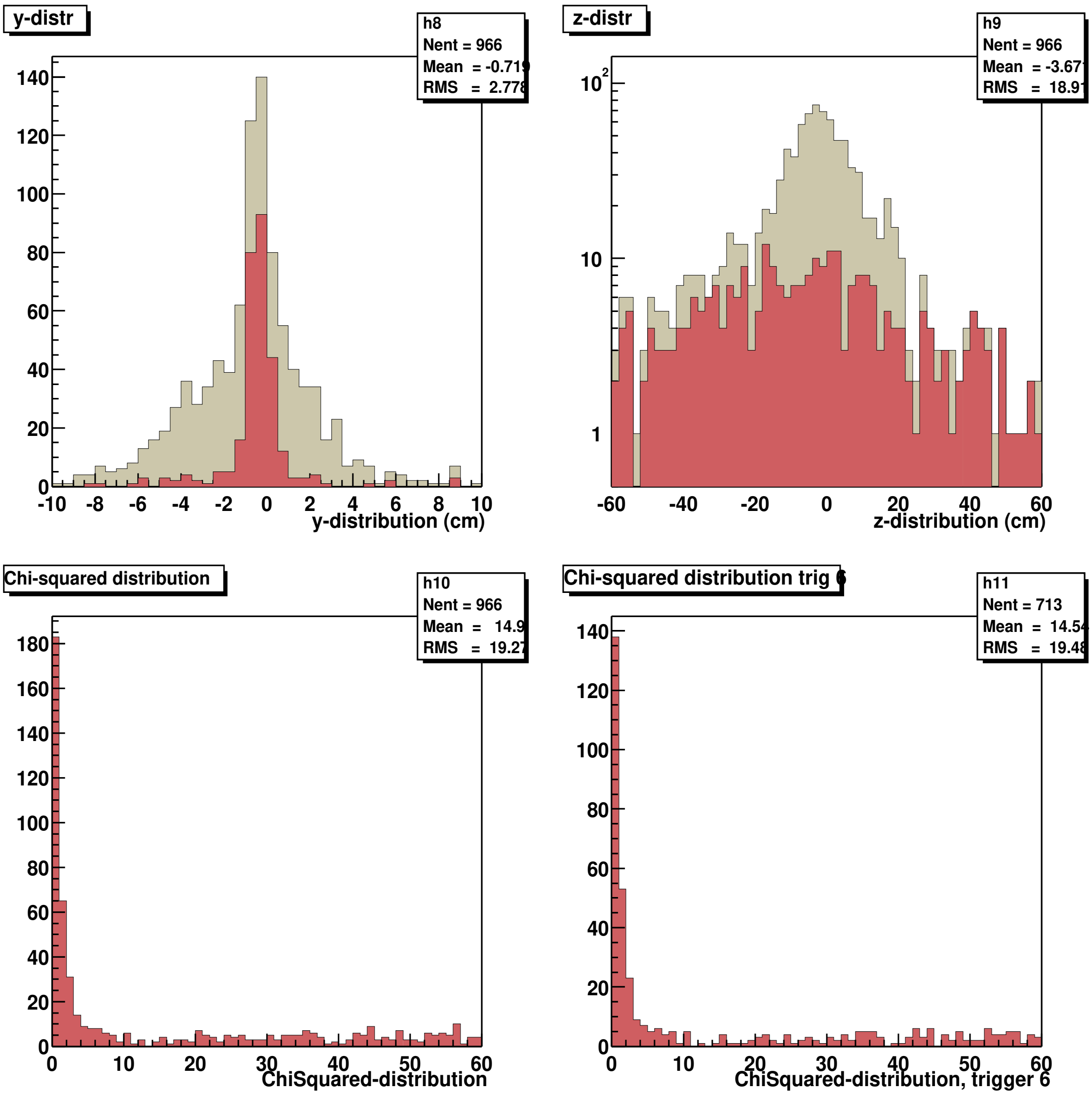

Figure 4: BrTPMTrackVertexModule-results from run 2235, seq 0.

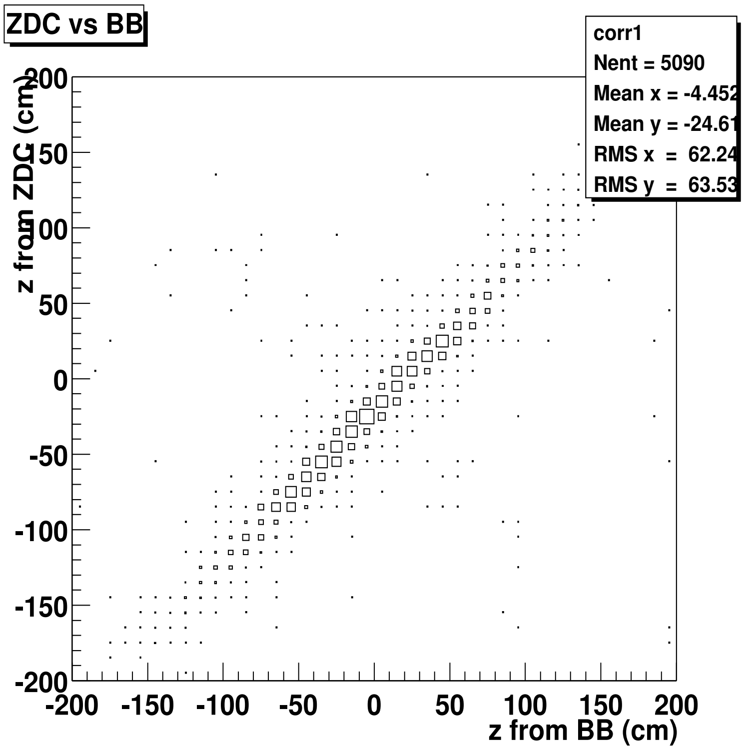

Figure 5: Correlation between BB and ZDC

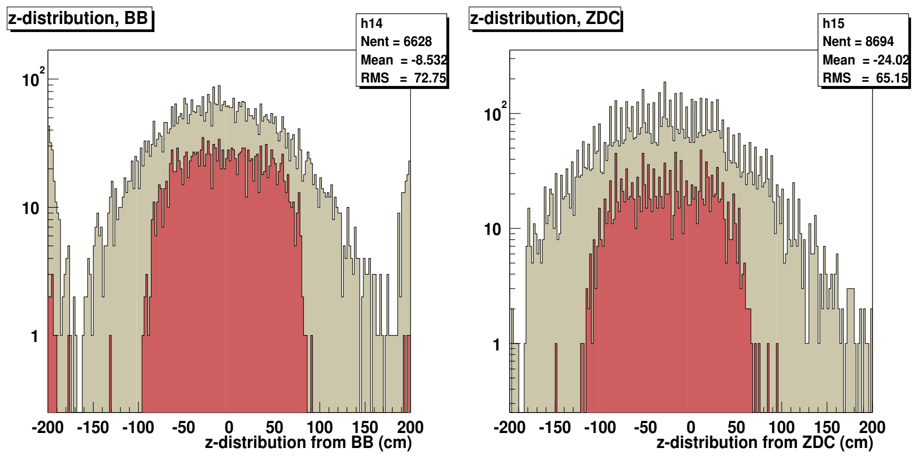

Figure 6: Z-distributions from the global detectors, run 2235 seq 000.

5.1.2 Z-distributions

For reference, I include in figure 6 the z-distributions from the global detectors for the run I have used in this note. The ZDC-distribution clearly shows a shift of roughly 30 cm.

5.2 BrTPMTrackVertexModule and BrTPMClusterVertexModule

Another important thing is of course that the two vertex-algorithms are consistent. Figure 7 shows the correlation between them.

The statistics are not too high, but the figure shows that under these constraints there is a very good correlation between the two modules. The fact that statistics here is lower than in the other pictures seems to indicate that the modules determine the vertex well for different events, but some of this is probably caused by the quite narrow cuts I have used.

5.3 TPM1 vs BB

In figure 8 I show the correlation between the TPC-vertex and the beam-beam-counters. On the left is data from BrTPMClusterVertexModule and on the right is BrTPMTrackVertexModule.

The correlation seems quite good for both modules. There may be a hint of a "double-band" structure in the first picture - I do not know if this is real or just an illusion.

5.4 TPM1 vs ZDC

Figure 9 shows the correlations between the TPC-vertex and the zero-degree calorimeters.

The correlation is as seen in section 5.1.2 shifted by roughly 30 cm (1 ns). It is maybe a bit noisy, but part of this has to do with low statistics. It is as

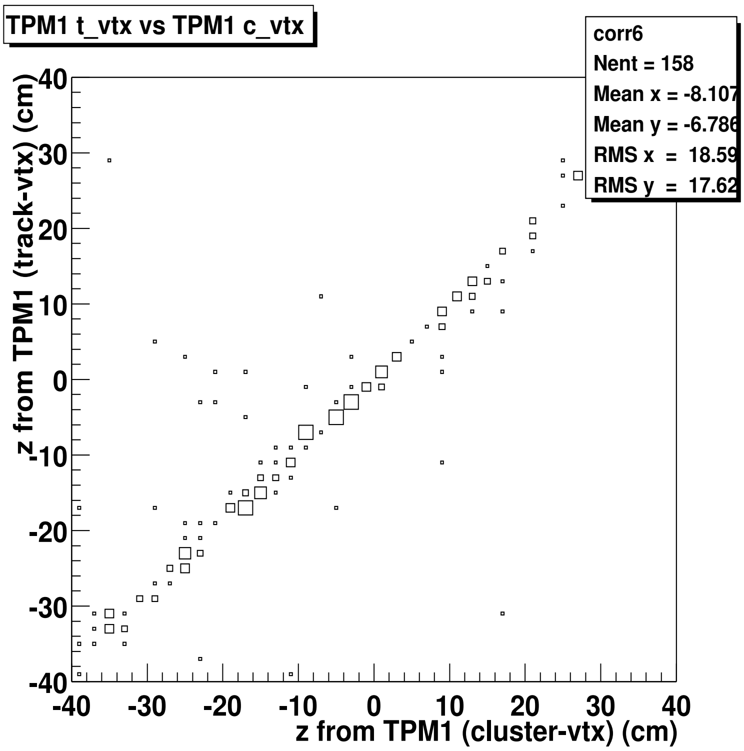

Figure 7: Correlation between the two vertexfinders

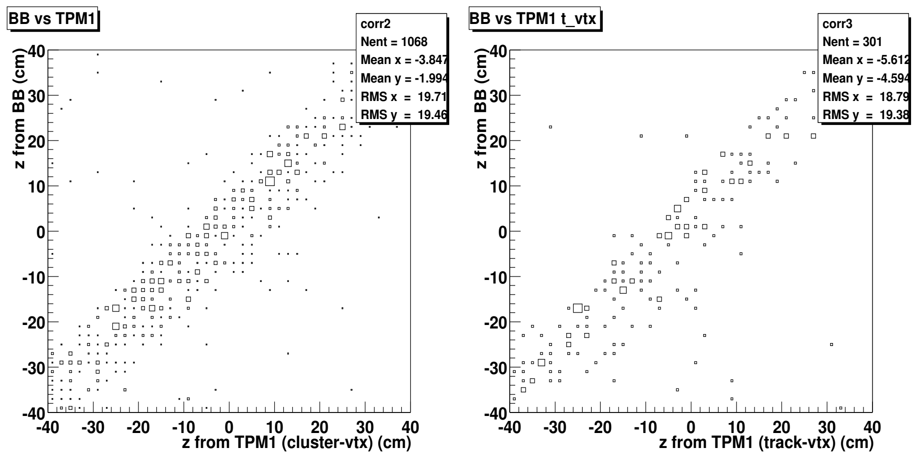

Figure 8: Correlation between TPM1 and BB

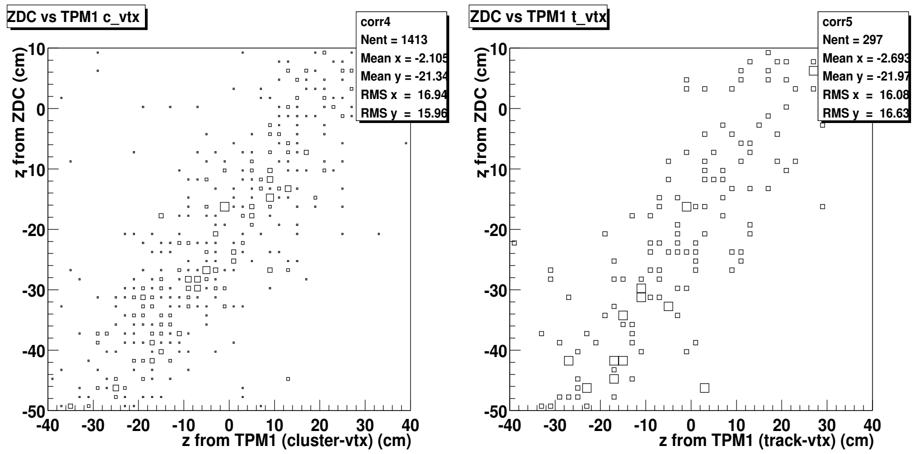

Figure 9: Correlation between TPM1 and ZDC

expected a bit broader than for the BB's, and I think that a clear band will emerge when I analyze more events.

6 Comments, issues and conclusions

All in all it seems that vertex-finding with TPM1 is working and giving consistent results with the global detectors. It may be that a rate of one good reconstruction per 5 trigger 6-events is too low - in that case more work needs to be done both on the modules themselves and on cleaning up the TPC (this should be done anyway...). The rate will also increase by almost a factor of 2 if we can use RMS $<$ 1cm as the cut-parameter. I leave this up to the PID-people, since they are the ones most in need of a precise vertex from TPM1.

One possibility which is not yet included is to use information from the BB's and ZDC's as input to the TPM1-modules. Now that we have shown that they are consistent, I think this can safely be done without introducing large systematic errors.

Another issue is the positioning of TPM1. We now seem to have control over the driftvelocity and possible edge-effects (if not the y-distributions would not have been centred at 0), but if it stands at a slight angle then this will influence both vertex-modules in a systematic way. This needs to be cleared up completely before we can use TPM1-results seriously.

In section 3.1 I mentioned that I remove a linear background from the histograms. I have just had pointed out to me that this may kill the skirts of a Gaussian, and thus maybe cause some of the problems I have had with the ROOT peak-fitter. I will look into this shortly.

Please send me any and all comments you have on the above results. If you wish to check any of this for yourself, the ROOT-file I have used to generate most of the pictures can be found on

www.fys.uio.no/~bjornhs/vtxnote/vtxnt_run002235.root