DAQ Software Documentation

Michael Phipps

Florida Institute of Technology, Melbourne, FL
mphipps2010@my.fit.edu

Abstract

This document is meant to provide a snapshot of the current state of software used in the data acquisition process in HEP Lab A – including data taking, monitoring, and analysis. It includes trouble shooting techniques for DATE, both in installation and application. It includes images and explanations of all raw data, pedestal, data suppression, monitoring and analysis plots. It will also thoroughly explain the concepts and structure behind the monitoring and analysis done in AMORE and its GUI. For more conceptual issues and questions on hardware, Mike Staib's thesis is more comprehensive, but this paper is meant to fill the hole in a more technical standpoint. To be clear, though this is meant to describe the DAQ system, it is not necessarily a "how-to" guide. For those sort of questions, please refer to the SRS How To Guide.

I. Scalable Readout System (SRS)

As the name suggests, the SRS was developed as a DAQ system that could be easily implemented across a small system of detectors or scaled up to a larger, more practical station. In the MTS' current form, the SRS follows the schematic shown in Figure 1.

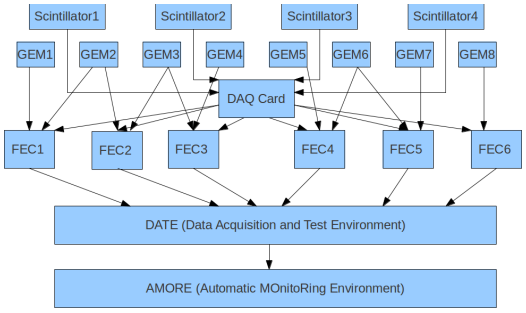

Figure 1: SRS schematic diagram

Eight (approximately) 300 mm GEM detectors (two top, two bottom, two left and two right) surround a central, active region. Plastic scintillator paddles connected to PMTs are placed on the outside of each detector pair and act as triggers. These triggers are synchronized by a DAQ card, and when the card receives a signal from two different scintillators within the same time interval, it signals each of six FPGA Front End Converters (FECs) to dump their data. The FECs are constantly receiving an analog signal from the 12 APV chips attached to each detector. They convert this data to a digital signal, store it and wait for instructions. When they receive a trigger, the appropriate data is released and filtered through one of two ethernet switches and routed to DATE and AMORE for processing, storage and analysis.

II. Data Acquisition and Testing Environment (DATE)

II.1 Slow Control: Initialize the SRS

Before each run, the active equipment (Network switches, FECs, ADCs, and APVs) must be properly configured. This occurs in two stages. First, as part of the Start of Run (SOR) processes, each SRS card is initialized. This includes setting the IPs and initializing the ADCs, FECs and APVs. These text files are found in the slow control folder in the home directory. The shell script is found in the same place, but the script is run automically through editDb (ie when "Start Processes" is clicked in the Run Control GUI). If you type "editDb" in the comand line, you will find this script referenced under the "Files" tab and as part of the

"SOR.commands". If you are adding or removing FECs from the previous implementation, you will have to edit this slow control shell script.

Of particular note, the FEC Cosmic Run text file is used to set the trigger sequence, the trigger delay and the number of time bins sent for each trigger. The register values can be found in the SRS Slow Control Manual in the slow control folder.

Now that the SRS is configured, the final step is to activate it. This is done at the start of a run by entering the command "StartRun" on the MTS PC. This command is set in the computer's ~/.bashrc file. If you are adding or removing FECs, make sure the shell script that this command references is activating the proper FECs.

II.2 Run Control Troubleshooting

Within the Run Control GUI, you may run into occassional trouble shooting issues. Problems tend to occur more during installation than during normal operations. The first three suggestions are for normal operations while the last one briefly addresses installation.

- Problem loading configuratino file: This warning is benign and does not affect run control. Disregard this error for now.
- Full disk: The raid is full and files need to be transferred to the cluster and deleted from the disk to open up storage space.
- Trigger mismatch: This error sometimes shows up in the DATE Infobrowser or the AMORE processing agent. If it does, stop the run, delete the .raw file from the disk and restart the run. This usually fixes the problem.
- editDb

  For most purposes, the DATE root directory should not be accessed directly. The editDb configuration GUI is sufficient for most configuration and installation purposes. To access this GUI, type "editDb" in the command terminal. Since we only utilize a small portion of DATE for the MTS, many of the settings in the GUI are superfluous. For a detailed account on configuring our implementation of DATE, see Mike Staib's documentation.

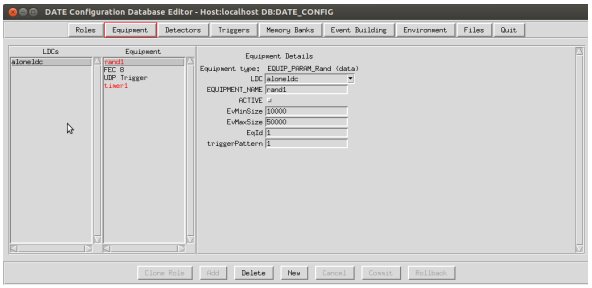

Figure 2: The editDb GUI

III. Automatic Monitoring Environment (AMORE)

AMORE was developed by the ALICE experiment as a monitoring platform meant to work in parallel with DATE . Because of its emphasis on monitoring, it was important to be able to perform analysis and visualization simultaneously, without interfering with or being dependent upon one or the other. To that end, they implemented a publisher/subscriber paradigm in which data is cyclically published to an independent MySQL database by one agent and subscribed to by another. In that way, the two sides of AMORE never directly interact, and visualization can proceed without interfering with analysis.

This rigid structure for AMORE is defined upon installation in the root directory /opt/amore/, while the customizable, MTS-specific analysis code is stored in the user directory ~/amoreSRS. Except for exceptional circumstances, the former should never be altered; all changes are made in the frontend amoreSRS directory.

Within the ~/amoreSRS/src directory, there are four modules: Common, Publisher, Subscriber and UI. Source code in the common directory is available to source within Publisher, Subscriber and UI, while source within one

of the latter three is restricted to other source within the same directory. This is quintisential to the publisher/subscriber paradigm.

In the current version of AMORE created for the MTS, we use the common and publisher folders extensively. We have never made use of the subscriber folder, and the ui folder has been largely deprecated by the external AMORE GUI described in section IV.

Though it is written in C/C++, there is no main function in AMORE. SRSPublisher is where all processes originate, and they are done in a cyclic manner, so data is published in the database after each cycle and made available to any subscribers. The structure of the publisher is shown in Figure 3.

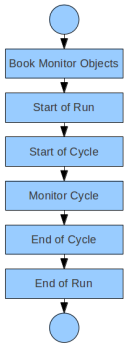

Figure 3: AMORE Publisher Routine

III.1 Mapping: Local and Global Strip Corrections

The strips on the detector are ordered sequentially, but by the time the data reaches the DAQ system it has undergone a number of transformations. For most analysis purposes, we are forced to develop mapping equations that untangle these transformations and bring us back to the original sequential order. The system we follow is shown and described in Figure 4.

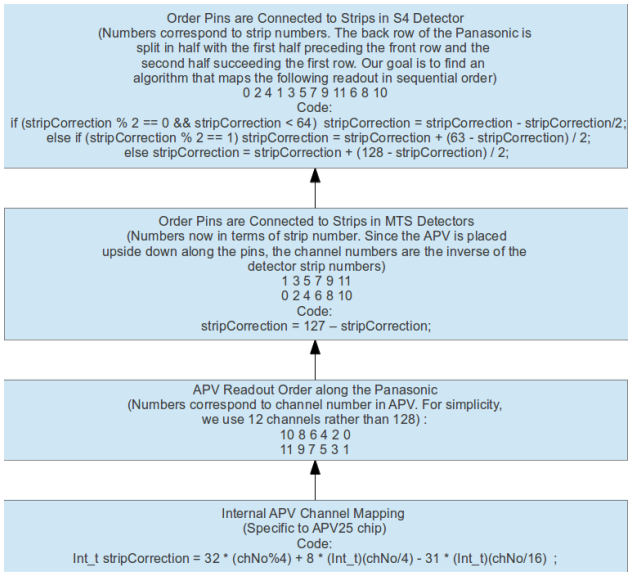

Figure 4: Local Mapping Corrections

When working backwards, the first correction we must account for is the internal mapping within the APV25 chip. The code for this transformation is shown in the image.

The APV is attached upside down with the 0-channel of the APV connected to the upper right pin of the Panasonic. The rest of the connections in a suppressed 12 channel format are shown in the second box of Figure 4.

As long as the X and Y planes of APVs are attached to the readout board in a right hand coordinate system, this order must be inverted. The 0-strip for the readout system in the MTS and the S4 is connected to the Panasonic pin in the bottom left corner. The rest of the arrangement and code is shown in the third box of Figure 4.

At this stage, the mapping for the MTS is complete and the data being read in the DAQ is returned to sequential order. In the S4, however, we have to make one last transformation. Instead of a back and forth readout pattern like the MTS, the first fourth of the strips are attached sequentially to the first half of the back row of pins. The next half of the strips are attached across the front row of pins and the final fourth of the strips are attached to the

second half of the back row. This pattern is shown in Figure 5.

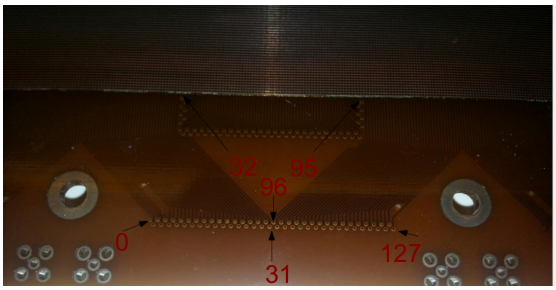

Figure 5: S4 Readout

To find the correct mapping, we used the simplified 12-channel readout shown in the third box of Figure 3 and opened up the back row manner described for the S4 readout. This results in the 12 digit pattern shown in the first box of Figure 4. The challenge was to find a set of equations that returned that series to sequential order. To do so, we split the pattern into three parts: the first three even digits, the middle six odd digits and the final three even digits. We then found the difference between each number and the number which we needed to map to. For example, 0 transformed to 0, giving a difference of 0; 2 transformed to 1 giving a difference of 1; 4 transformed to 2 giving a difference of 2. In other words, the difference increases by 1 for each strip in this group. The first if-statement shown in Figure 4, was developed to fit this pattern.

The final challenge for the S4 was correcting for the left hand coordinate arrangement of the X and Y APVs. This caused the readout for the two detector planes to be inversions of each other. To correct this, one final 127 - strip-Correction tranformation is made for all y APV channels. The enitre code for this mapping can be found in the SRSAPVEvent.cxx file.

At this point, the local mapping is complete. All strips within each APV are once again arranged sequentially. All that is left is to find a global mapping that arranges each APV and the strips it contains sequentially on each detector plane. In the case of the MTS and S4, each detector plane has six APVs. The order of each APV is designated in the mapping configuration file, and the strips of each APV are globally corrected by adding $128 * APVnumber$ to each strip. This mapping is applied within the SRSHit.h file.

III.2 FEC Event Decoding

No matter what we plan to do in AMORE, the first task always involves mapping and storing our raw data. Subsequent processes will then use this raw data for monitoring or analysis purposes.

At the beginning of each monitoring cycle in SRSPublisher, the raw data from one complete event across all FECs and APVs is retrieved from MySQL and parsed using the DATE object, TDATEEventParser. This breaks the event into individual units of local data collection (LDC). The MTS, however, uses only a single LDC, so this step is trivial. We then use the EventParser to divide the LDC into six separate components, one for each FEC. Each FEC is referred to by a unique eqID variable. We then cycle through each FEC and store its raw data in a buffer. The buffer and the size of the buffer are then used to instantiate an SRSFECEventDecoder object for each FEC.

The event decoder constructor uses the mapping file to initialize 16 APVs in each FEC. We then loop through the raw data buffer for each FEC. The discrete data in this buffer is called a word and each APV contains 503 words, leaving each FEC 8049 words (with one word added by DATE as an event counter). The first check we make is whether we have indeed reached the final word of the buffer. If we have, we construct the event from the last APV, delete the final event counting word, and exit the loop. If we have not, we then check whether this is the beginning of a new packet. A packet means all data for a particular APV. At the beginning of each packet, we build the apvEvent for the previous APV, unless this is the first apv in the FEC, in which case we initialize the event, clear our data container (called data32BitsVector) and continue to the next word in the packet. The next 500 words

in the packet are all raw data from one particular APV from one particular event. Each of these words is cycled through and added to data32BitsVector. At the end of the package, we move on to the next APV, intiatialize an SRSAPVEvent object and group all 500 words under that object. Each SRSAPVEvent for a particular event from a particular FEC is added to the list fFECEvent in the SRSFECEventDecoder object. Each SRSFECEventDecoder is then added as a FECEvent in an SRSEventBuilder object. The EventBuilder object will be used by monitoring and analysis processes to extract raw data.

In order to ensure our FECEvents are properly ordered, the event number for the first FEC in any event is stored under the variable "fTriggerCount". As we cycle through all six FECs, this variable is compared to the event number of each FEC. If they do not match, then we have a trigger mismatch and AMORE throws an exception.

III.3 Raw Data

If raw data histograms are uncommented within the designated histogram configuration file, we can add the decoded data from each FEC to raw data histograms for each APV. This is done within the FillRawDataHistos() method in the SRSHistoManager object. The eventbuilder object described above is passed to the SRSHistoManager and the histograms for all 96 APVs. An SRSAPVEvent object is declared for each APV and the proper raw data for that APV is extracted from eventbuilder. This data is still in 32 bit vector format and needs to be converted to a 16 bit vector format. This is done through the following hexadecimal bitwise conversion:

```
for (rawData_itr = fRawData32bits.begin(); rawDat_itr != fRawData32bits.end(); ++rawData_itr) {
UInt_t word32bit = * rawData_itr ;
if (((word32bit » 8) & 0xffffff) != 0x414443){
UInt_t data1 = (word32bit» 24) & 0xff ;
UInt_t data2 = (word32bit » 16) & 0xff ;
UInt_t data3 = (word32bit » 8) & 0xff ;
UInt_t data4 = word32bit & 0xff ;
fRawData16bits.push_back(((data2 « 8) | data1)) ;
fRawData16bits.push_back(((data4 « 8) | data3)) ;
}
}
```

To perform this algorithm, note the following conversions

$$x >> y \equiv x/2^y \tag{1}$$

$$x << y \equiv x * 2^y. \tag{2}$$

For an example illustrating the rest of this bitwise hexadecimal calculation, see the ComputeRawData16bits method within the SRSAPVEevent.

After performing the conversion, each 32 bit vector entry provides two entries to the 16 bit vector. Since each 32 bit vector for each APV contains 500 entries, each 16 bit vector contains 1000 entries. Each entry corresponds to a discrete voltage value provided by the ADC. And each vector contains six separate time bins for each channel in the APV. No mapping has been done yet, so the strips are not displayed consecutively within each time bin.

Raw data histograms are configured with 1000 bins and each bin is filled with the appropriate 16 bit raw data entry. The final result is shown bleow in Figure 7.

III.4 Raw Pedestals

Plots produced for raw pedestal and pedestal processes are slightly distinct from the other monitoring plots because the root files of these histograms are used in analysis for pedestal subtraction. However, like the other monitoring plots, they may also be published and monitored within the AMORE GUI.

Raw Pedestal plots are filled, one event at a time, within the SRSRawPedestal object. The entire event is retrieved from eventbuilder and stored in a TList. Each entry corresponds to an APV and the data is stored in an SRSAPVEvent. Each of the 96 APVs are then cycled through. The header level is set from the mapping file

and the raw pedestal data for each APV is calculated. The first step is to convert the APV raw data from a 32 bit vector to a 16 bit vector. This is done in the same way as described in the FEC Event Decoder.

Next, the time bin common mode is computed for the particular APV. The 1000 words in the fRawData16bits vector are cycled through until three consecutive words are found below the given header level. When that condition is met, it means we have reached the dividers that separate each time bin. We jump forward nine words and begin processing meaningful data that corresponds to particular strips.

Each APV has 128 channels, but when compared to the 128 strips of the detector, the channels are misarranged. To return our data to recognizable form, we must map each APV channel to its corresponding strip. This is known as the local strip correction, since it rearranges the strips within each APV but not each APV within each detector plane. Each channel is then cycled through and the appropriate correction is made. The raw data for each strip is inverted by subtracting from 4096. In the raw data plots, an event is marked by a voltage drop. In all subsequent plots, an event will appear as a jump in ADC count. For raw pedestal processes, the common mode offset is set to 0. All that happens at this point is the mapping of channels to strips and the inversion of the raw data.

Then we return to ComputeRawPedestalData() and cycle through each time bin for each strip (post-mapping). The raw data in each time bin is aggregated for each strip and then divided by the number of time bins to find the average raw data value in time for each particular strip. This value is the raw pedestal for an individual strip across each time bin across each of the 5,000 sampled events.

We then return to SRSRawPedestal and cycle through the raw pedestals for each strip and fill the raw pedestal histograms for each APV with the appropriate raw data at each strip. The RMS and Mean plots are not created from this data until the pedestal process, but their values at each strip are taken from this raw pedestal plot.

III.5 Pedestals

The pedestal process is designed to work in a similar manner to the raw pedestal process. We again work with one event at a time, this time within the SRSPedestal object. The data from each APV is assigned to an SRSAPVEvent. That apvEvent then uses the 1d raw pedestal histograms previously created and uses a ROOT method to solve for the RMS (noise) and mean (offset) of each plot. The header level is then set from the mapping file and pedestal data is computed.

Again, the first step is to convert the raw data from a 32 bit format to 16 bit. Then the common mode values are calculated. For pedestals, this again involves cycling through each channel, finding the appropriate strip correction and inverting the raw data values at each strip. The difference between this and the raw pedestal process comes in substracting the raw pedestal mean for each strip from the raw data value at that strip. This value is set equal to the total common mode offset and as each strip is cycled through, the corresponding offset is added to the total common mode offset count. This is then divided by the total number of strips to find the common mode offset. Each time bin of each strip is then cycled through and the common mode offset is subtracted from the raw data at that particular strip in time. This value is aggregated for each of the six time bins and then divided by six to find the pedestal value for that strip.

We then return to SRSPedestal and fill the histogram for this particular APV with the pedestal value of each strip. Again, the RMS and Mean plots are not created from this histogram until the next process.

III.6 Analysis

The analysis section is specific to the MTS or any station design that requires track selection and more complicated event analysis.

We begin in SRSTrackBuilder::BuildDetHits and cycle through each of the 8 GEMs and, in turn, each plane for each GEM for a single event. An SRSDetectorPlaneEvent is assigned for each detector plane and composed of a list of APV events from each APV on the detector plane. Each SRSAPVEvent is extracted from the eventBuilder described in the FEC decoder section. The location of the particular APV on the plane, its header level and number of connectors are retrieved from the mapping configuration file and assigned to the APV event.

Each of the six APV events must then undergo the different methods of data suppression in order to isolate the event in time and space and eliminate spurious data. The mean and RMS pedestal data for each APV is retrieved and assigned to the corresponding APV event. We then convert the raw data from the FECs from 32 to 16 bits. Similar to the raw pedestal and pedestal processes, the time bin common mode is then computed, cycling through each channel of APV raw data, performing the strip corrections and inverting the raw data at each strip. Unlike the previous two processes, the total common mode offset is equal to the raw data at each strip minus the pedestal mean of the strip, aggregated across all strips in the APV event. This value is then divided by the total number of channels to form the common mode offset for a particular time bin.

We then cycle through the 128 strips and the 6 time bins for each strip and perform common mode suppression. This is done by subtracting the common mode offset at each time bin from the raw data of each strip. We then take the pedestal subtraction for each strip by subtracting the pedestal mean at each strip from the raw data of each strip. A cut is designated as the sigma level multiplied by the RMS at each strip. The resulting raw data values at each time bin are added to a vector where the mean charge, max charge and max time bin are calculated for each strip. The charge for the strip is then initially assigned as 0 minus the baseline mean found from pedestal data. If the mean charge across a strip is less than the cut ($sigma * pedestalRMS$) then the charge at that strip is assigned as the mean charge across the time bins. If the mean charge is greater than the cut, charge is assigned the max charge. We perform zero suppression by checking if the resulting charge is less than the cut. If it is, the raw data at that strip is cleared, while those strips that surpass the cut are designated as hits and assigned an SRSHit object. As the data from each APV is calculated, the hits are stored, while the noise is suppressed.

Global mapping is done at this time to map the location of the hit within the inidividual APV to its actual location along the detector plane. This is done in the method SRSHit::SetStripNo() and described above in the mapping section.

After data from all APVs in the detector plane have been analyzed, the hits along each detector plane are cycled through and clusters of strips are formed. A single cluster is defined as consecutive strips that register hits. If the size of the cluster is greater than 30 strips, the hit is suppressed and cleared. Please note, this is somewhat arbitrary and should be optimized for each station.

The position of the cluster is found in SRSCluster.cxx through the following routine:

```
for (int i = 0; i < nbofhits; i++) {
q = ((SRSHit*)temp[i])->GetCharges();
hitposition = ((SRSHit*)temp[i])->GetPosition();
fcharges += q;
fposition += q * hitposition;
}
fposition /= fcharges;
```

All clusters in the detector plane are then cycled through to check their quality. This is done by graphing the position and charge of each hit in the cluster and fitting it to a Gaussian curve. The position of the cluster is then adjusted and assigned as the mean of the fit if the calculated cluster position lies within 0.4 of the mean of the fit. The reduced χ^2 of each cluster is also calculated and recorded in the cluster list.

After performing these calculations for each

detector in the detector plane, we return to SRSTrackBuilder and assign all detector plane clusters to an SRSDetectorEvent. After doing this process for each of the two detector planes (X and Y), we check if a hit is registered on both planes. If it is, we create a station event and add the detector event. This process is then performed for each of the 8 detectors. We then qualify an event for track selection if both detectors from exactly two stations (top, bottom, left or right) record a hit.

If the event qualifies for track selection, we start by forming all possible track segments. We cycle through each station event in SRSStationEvent::MakeAllHits(). Within this method, we cycle through each of the detectors in the event and assign that detector event to the SRSDetectorEvent object and create a list with all hits on the detector. The position and charge for each hit is recorded and the global coordinates of the hit are calculated based off the detector locations provided in the mapping configuration files. This means the hit is localized on a particular strip but then mapped to a particular location in space based off the coordinates and rotations provided in the configuration file and calculated through an alignment script. The charge and location is then assigned and stored within an SRSDetectorHit object which is stored in each SRSDetectorEvent.

We are still dealing with one particular pair of detectors (top, bottom, left or right) and to form the various track candidates we cycle through the detector hits for each of the two detectors, pivoting off one hit from one detector, such that every hit on (for example) the first top detector is compared to every hit on the second top detector. Each pair of hits is designated and recorded in an SRSTrackCandidate.

Now, we can start track selection in SRSTrackBuilder::FindTrack() by comparing the track candidates in each of the two SRSStationEvents (top, bottom, left, or right). The distance of closest approach (DOCA) threshold is assigned as 10 and the minimum DOCA as an arbitrarily high 3000. The direction in space of each of the track candidates is calculated (using the global position of each of the hits described above) and the angle between these tracks is found using the ROOT library TVector3::Angle() method and is defined as the scattering angle of the track. To make ensure this function returned the correct angle, we check if the angle is greater than 180 - itself. If it is, the angle is set to 180 - itself, giving us the smaller of the two angles. The charge of each track and the total charge of the two are recorded, and the charge sharing is found by taking the total charge in Y of both track candidates divided by the total charge in X of both candidates. The poca point for the track candidates are then found by the following routine:

```
hit1 = trackCandidate1->GetFirstHitLocation();
hit2 = trackCandidate1->GetLastHitLocation();
hit3 = trackCandidate2->GetFirstHitLocation();
hit4 = trackCandidate2->GetLastHitLocation();

const TVector3 U = hit2 - hit1;
const TVector3 V = hit4 - hit3;
const TVector3 W = hit1 - hit3;

Double_t a = U.Dot(U) ;
Double_t b = U.Dot(V) ;
Double_t c = V.Dot(V) ;
Double_t d = U.Dot(W) ;
Double_t e = V.Dot(W) ;
Double_t D = (a*c)-(b*b) ;
Double_t sc, tc ;

if (D < 0.001) { // Lines are almost parallel
sc = 0.0 ;
tc = (b>c ? d/b : e/c) ;
}
else {
sc = ((b*e) - (c*d)) / D ;
tc = ((a*e) - (b*d)) / D ;
}

TVector3 Psc = sc*U ;
TVector3 Ptc = tc*V ;
TVector3 subdP = Psc - Ptc ;
TVector3 dP = W + subdP ;
```

```
TVector3 PSc = hit1 + Psc ;
TVector3 PTc = hit3 + Ptc ;
TVector3 midPsc = 0.5*PSc ;
TVector3 midPtc = 0.5*PTc ;
TVector3 pocaPoint = midPtc + midPsc ;
doca = dP.Mag();
return pocaPoint;
```

If the resulting DOCA is less than the minimum DOCA (defined earlier as 3000) then the minimum DOCA is set to DOCA. The track selection criterion is then the following:

1. The pair of SRSTrackCandidates (four total SRSDetectorHits) from any given SRSStationEvent with the shortest distance of closest approach (DOCA)

2. The POCA point for that track must be greater than 10 mm from the nearest detector hit.

III.7 Tracking

Tracking runs are those that produce monitoring plots. They can be performed either online or offline. SRSHistoMananager is the primary class for these runs. They do not employ track selection, and therefore can be used on the MTS or single detectors like the S4 for detector characterization.

We have three groups of tracking plots: zero suppression, cluster statistics and 2d plots. The type of plots produced are set within the histo config files in the ~/amoreSRS/configFileDir directory. Look at the appropriate shell script and determine which histogram config file you are using. Then comment or uncomment the appropriate lines in the configuration file to produce a particular type of histograms in your next run.

Histograms are initially configured and set aside for publishing at the beginning of each run in the Book Monitor Objects portion of SRSPublisher. This cycle appears just once each run and precedes the Monitor Objects cycle. Like the previous processes, during tracking runs raw data is decoded by the FEC Event Decoder, transformed from 32 bit to 16 bit vectors, inverted, and subjected to common mode suppression, pedestal subtraction and zero suppression.

At that point, the various histograms are filled as defined in SRSHistoManager. Suppression plots are made to display a single event and show the evolution of data suppression cuts made to the data. Cluster and 2d plots are cumulative and provide large scale detecter characterization. Examples and explanations of all of these plots are shown below in the monitoring section.

III.8 Alignment

This section should be expanded upon in future documentation. It includes the alignment routine in AMORE and the offline alignment script found in the same location as the offline POCA analysis (~/workspace/src).

IV. AMORE GUI

The AMORE GUI – accessible by the "amoreGui" command – is the generic GUI developed by the team at ALICE to work in conjunction with but external from AMORE. It appears on the subscriber side of the publisher/subscriber paradigm but its code can be changed without affecting a current AMORE process. This is in contrast to the UI module that is found directly in the amoreSRS directory. Any GUIs built in that directory are more closely linked to the publisher and both the publisher and the GUI have to be recompiled any time changes are made.

In the past, we used crude GUIs that can be found in the UI module. These could still be used but they are more heavily hard coded and, in their current form, require separate programs for each agent being used. For example, if you are using the agent SRS01, you would have to use the program SRSUI. If you are using the agent SRS02, you would use the program SRSUI2. You also have to suscribe directly to a particular histogram in order to have access to it in the GUI. This means you

would have to know and input the name of the file ahead of time. The user also has little to no freedom to change the layout of these GUIs.

The generic GUI (accessible by the command "amoreGui") is much more dynamic and overcomes all of these problems. It allows us to visualize histograms beings published by any agent at any time. We could even look at multiple histograms from multiple agents at the same time and have the freedom to organize the plots on our canvas in any way we choose. To access a particular plot, you must first create a canvas or have a previous one opened. You can create a new canvas by clicking the "new canvas" icon or you can open an existing canvas layout by clicking the "open folder" icon. Once you have done this, you can access particular plots by opening the drop down menu on the left hand side of the GUI. You will then see all plots being published by the AMORE agent. Double click the name and the plot will appear on your canvas. We can access the root file of a particular plot by double clicking that plot on the canvas. We can then edit or save the plot in a manner similar to a TBrowser (In ROOT, a TBrowser is accessible by logging into ROOT with the command "root -l" and then typing "new TBrowser()". You are then able to view, modify or save any files saved with a ".root" extension.)

The source for this GUI is found in ~/dqmv.

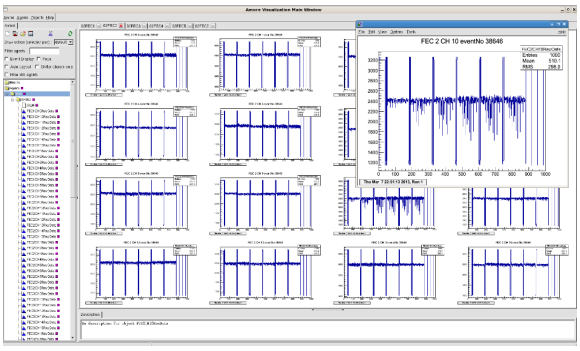

Figure 6: amoreGui

V. MySQL

Both DATE and AMORE use a MySQL database to store data before writing to a disk. In DATE, this is implemented by the newMysql.sh script and the subsequent commands "./date/setup.sh" and "newDateSite.sh". In AMORE, this is implemented by the "amoreMysqlSetup" command. This is all demonstrated in Mike Staib's Final Documentation.

In general, you should not have to interact directly with MySQL, as most of this is automated and done behind the scenes. However, if you are updating or reinstalling DATE/AMORE and encounter errors in the procedure described in the documentation, you may need to manually remove already existing MySQL tables.

To do this, open a command line and log into the MySQL server by typing "mysql -u root -p". When it asks for a password leave it empty and press enter. You can then see the databases on the system by typing "SHOW DATABASES;". Identify the correct database and type "DROP DATABASE <Database Name>;". Be careful using this command because you will lose any information stored in the old database.

You also may occassionaly receive an error from AMORE stating that a particular agent is dead. If that is the case and restarting the machine does not resolve the problem, you need to manually delete the agent from MySQL and then recreate it using the "newAmoreAgent" command described in the documentation. To delete this agent, log into MySQL, type "SHOW DATABASES;" to see the list of databases. Identify the correct database and type "USE AMORE;". Then type "SHOW TABLES;". You should now see a list of all the configured agents. Find the one you wish to delete and type "DROP TABLE <Table Name>;".

VI. Monitoring

VI.1 Raw Data Plots

Raw data plots depict a single APV event before any form of strip correction, data suppression or tracking. In figure 7, the event is shown in terms of a 16 bit data index that is only one step removed from the 32 bit data format in which it arrives from the FEC. Each event is separated into six 25 ns time bins (as set by the slow control FEC configuration file). Within each time bin the raw data from each detector strip is ordered in terms of the channel number within the APV. As such, the hits within each time bin do not appear sequentially in these plots. What they do illustrate is the time bin of maximum charge, giving us our best estimate in time for our event. If our slow control settings are properly configured, events should generally peak in the middle time bins.

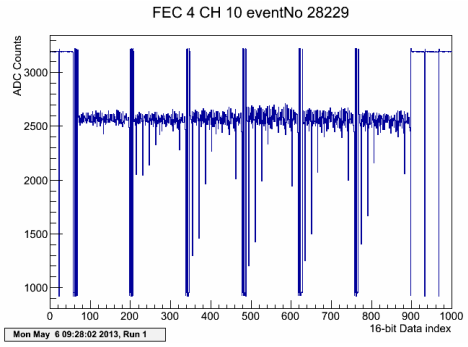

Figure 7: Raw Data Plot

Troubleshooting:

- Time bins show up empty or partially empty – This often happens in the first run after changing scenarios and is usually indicative of a bad connection between the HDMI cable and the APV. Use the labels on the monitoring plot to identify the offending FEC and HDMI cable. Then use the label on that cable to identify which APV it belongs to. If it's from one of the APVs by the station opening, the cable was likely bumped while changing the station. Make sure the connection between the APV and HDMI is secure and very gently nudge the cable back into place. Then shut off the power to the corresponding FEC (waiting until all lights inside the FEC are off), restart the power and see if the problem is fixed.
- The channel shows up with excessive noise – This is usually indicative of a bad connection between the HDMI cable and the FEC. Unplug the FEC (waiting for all of its lights to shut off) and then remove and replug the HDMI cable that corresponds to the faulty channel. Restart the power to the FEC and see if the problem is fixed. If this does not work after a few tries, you may need to replace the HDMI cable.

VI.2 Pedestal Plots

In order to properly suppress pedestal data from each data run, we must perform two processes: raw pedestals and pedestals. Each of these in turn publish two types of histograms per APV: mean and RMS histograms. The pedestal mean is used to localize each event in space, while the RMS is used to suppress noise in each channel.

[This section edited by MH] Pedestal data is taken at an HV where there is no gas gain in the detector (typically 2000V). The voltage baseline of each strip is sampled 5000 times. If the detector were perfect and without any noise, we'd digitize the same analog value 5000 times, which would result in a distribution of 5000 identical digitized values that would give a distribtuion that basically resembles a zero-width delta function. With analog noise present on the strips that distribution widens. The mean of the pedestal distribution is a measure of the mean analog voltage, i.e. an offset from 0V or the "pedestal" on top of which a signal pulse sits. The rms-width of the pedestal distribution is a measure of the noise in the channel. The more noise is

present, the more the sampled baseline values will fluctuate and the wider the pedestal distribution will be.

In raw pedestal plots, APV channels have undergone strip correction and the raw data in each channel has been inverted, giving us a different measurement from the raw data plots. Each time bin for each strip is cycled through and the common mode offset – defined as 0 for raw pedestal runs – is subtracted from each strip in time. Once each time bin is cycled through, the raw data for a particular strip across each time bin are added and divided by the number of time bins to find the raw pedestal value for each strip. The mean and RMS of these values across all 128 strips is what appear in Figures 8 and 9.

- Raw Pedestal Mean

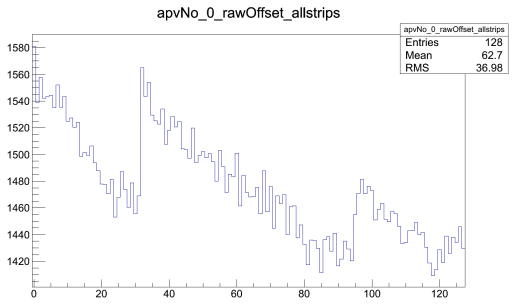

Figure 8: S4 Raw Pedestal Mean Plot

- Raw Pedestal RMS

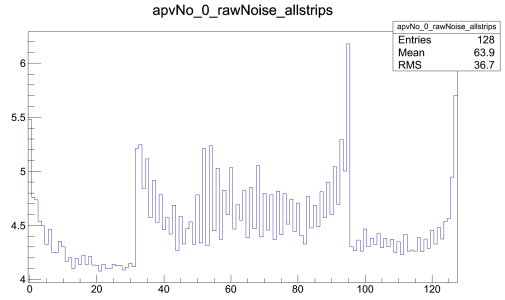

Figure 9: S4 Raw Pedestal RMS Plot

The only difference between the Raw Pedestal and Pedestal plots is that the common mode offset is no longer zero. It is now defined as the average raw data across all strip minus the raw pedestal offsets at each strip. This value is subtracted from each strip in each time bin and the average raw data across each time bin is found for each strip. This data is plotted and the mean and RMS of this plot are used to produce Figures 9 and 10.

- Pedestal Mean

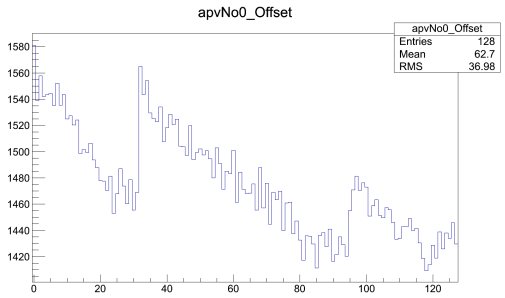

Figure 10: S4 Pedestal Mean Plot

- Pedestal RMS

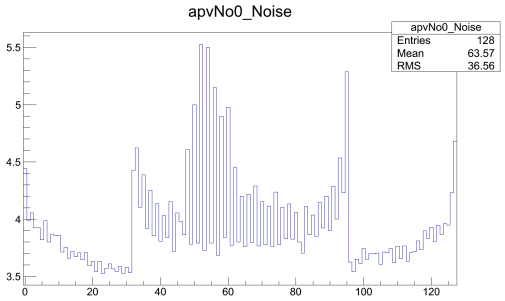

Figure 11: S4 Pedestal RMS Plot

VI.3 Suppression Plots

The following six plots were taken from the MTS and depict a single event. The procession of cuts made to the original data is evident from one plot to the next. By the end of the process, we will have

isolated each event at a particular location and time and suppressed all unnecessary data.

- Raw Plane Data

  When an event is detected and triggered by the scintillators, the FECs dump all data stored for the corresponding detectors from the appropriate time bins. This process is controlled by DATE and the slow control configurations sent to the SRS. When the data is ready for processing it is sent to AMORE and looks like the data shown in Figure 12 when grouped as a single time bin across a single plane of each detector.

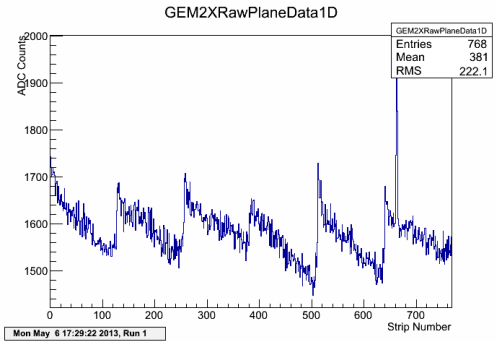

Figure 12: Raw Plane Data PLot

- Common Mode Suppression

  Common mode suppression refers to the event normalization across each time bin. Each event is generally configured with 25 ns time bins, and, in the MTS, each event is defined as six time bins. Since there is baseline charge fluctuation from time bin to time bin, common mode suppression is necessary to normalize and isolate the event at its time bin of maximal charge.

  The difference between Figures 12 and 13 is subtle, and the way it is suppressed is not completely clear from these plots alone. That is because the suppression made to each strip in Figure 13 is independent of the other strips shown. It is only dependent on the ADC count for the same strip number at different time bins. This information is suppressed from these plots but an example of the ADC count of the hit at strip 670 can be seen in the Figure 15. For this reason, the mean across all strips in Figures 12 and 13 changes by only 0.2 ADC counts.

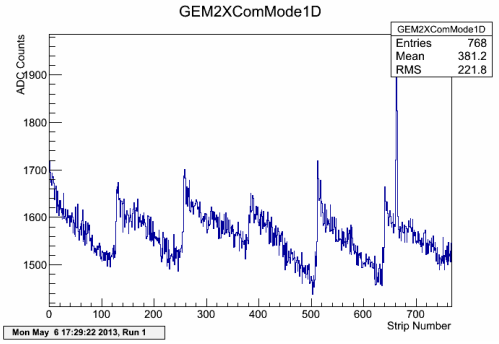

Figure 13: Common Mode Suppression Plot

- Pedestal Subtraction

  Pedestal subtraction refers to the stage at which previously generated background pedestal plots, like those shown in Figures 8-11, are used to suppress noise, normalize the ADC count across each APV, and localize each hit in space. This eliminates the downward sloping ADC pattern across each APV and suppresses the absolute noise.

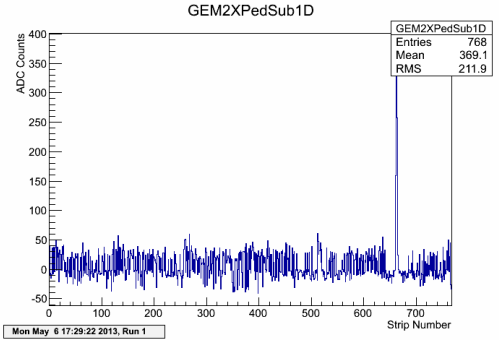

Figure 14: Pedestal Subtraction Plot

- Zero Suppression

Zero suppression acts next to completely isolate an event and suppress the remaining noise. As Figure 15 shows, this greatly reduces the data flow. Instead of storing baseline fluctuations from 765 channels of noise (in the event shown in this plot), we reduce those bins to zero and adjust the three containing our event accordingly.

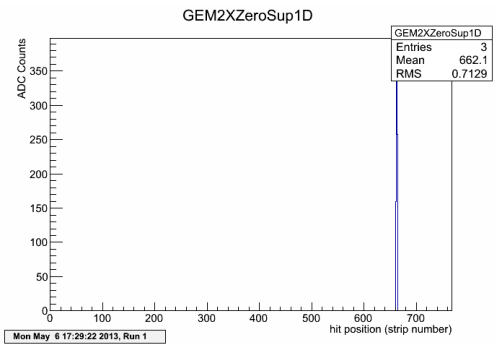

Figure 15: Zero Suppression Plot

- Plane Time Bin

  This 2d histogram shows the evolution of our event over multiple time bins after suppressing noise and normalizing across each channel and time bin. For a properly reconstructed event, like the one shown, we see a wave like shape that peaks in a particular time bin. The final cut is then made to suppress all non-maximal charge time bins.

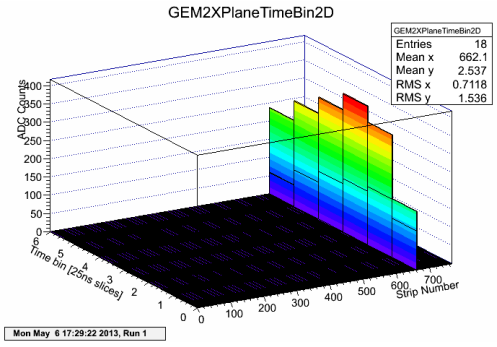

Figure 16: Plane Time Bin Plot

- Time Bin with Max Charge

  This 2d histogram shows the effect of the final cut across the time bins. After undergoing this suppression process, we now have an event that is localized in both time and space and have eliminated background noise.

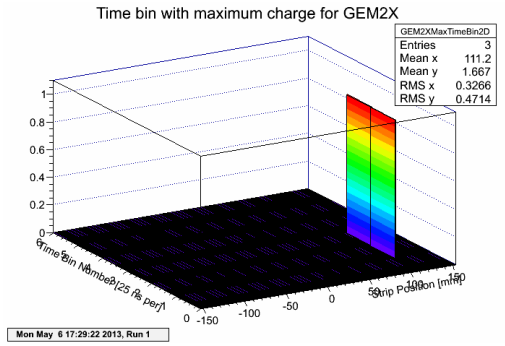

Figure 17: Plane Time Bin with Max Charge Plot

VI.4 Monitoring Plots

The following monitoring plots are designed to be viewed in real time as we take data, allowing a check on the integrity of our results. All the data in these plots first undergoes the suppression techniques described above, and as such, you must take and process a pedestal file before you can view these plots. But note, only the event display uses data from more rigorous track selection.

- Cluster Size

  When an ionization event occurs multiple strips in both dimensions register the charge. These strips are referred to as clusters, and the cluster size is the total number of consecutive strips in one dimension showing noise above background levels. This is shown in plot 18.

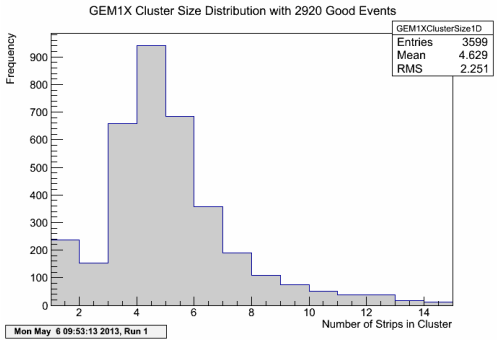

Figure 18: Cluster Size Plot

The smaller the cluster size, the better the spatial resolution of our detectors. Because cosmic ionization events typically come from a 45 degree cone centered above the station, tracks transversing the top and bottom detectors occur at smaller azimuthal angles (taken from the z dimension) and result in smaller cluster sizes than those entering the side detectors. These plots can be seen as a test of our ability to localize a particular event.

The number of good events in these plots refers to the number of differentiable events, while the number of entries refers to the number of good events times the mean cluster multiplicity. So all clusters appear as entries, even those that occur within the same time bin and are indiscernible from one another.

- Cluster Multiplicity

  Whereas the cluster size refers to the number of strips activated during a particular event, the cluster multiplicity refers to the number of clusters found at any given time within a single plane of a single detector. This may be indicative of multiple events being registered or of sporadic noise. Regardless, a high cluster multiplicity interferes with our ability to properly reconstruct a track by presenting multiple routes by which the ionizing particle may have traveled.

Note that the number of good events in these plots is equal to the total number of entries.

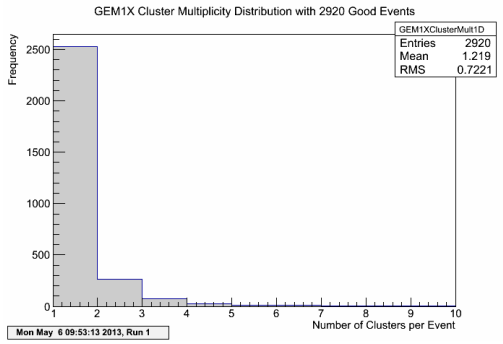

Figure 19: Cluster Multiplicity Plot

- Hit Distribution

  The hit distribution refers to the number of times any particular strip across the detector has detected an event over the course of the ongoing data run. Over a large enough sample size, we would expect to see this distribution to flatten out across all strips. If it does not, then the geometry of the station or the hardware being used is biasing our results. In Figure 20, the two empty strips occuring at about $\pm$ 50 correspond to the spacers used in each detector.

  In these plots, the number of entries corresponds to the number of strips in each cluster of hits across all events. In other words, it can be seen as an aggregate view of the data shown in the cluster size and multiplicity plots.

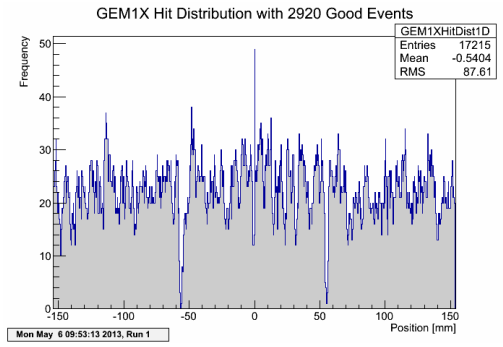

Figure 20: Hit Distribution Plot

- Time Distribution

  A time bin distribution plot shows the distribution of time bins in which the maximum charge occurs. These plots can be used to adjust the slow control settings to either suppress the number of time bins or to adjust the window of the event in relation to the trigger. Ideally, the max time bin should be in the middle of this plot. That would indicate that the triggers are configured correctly and the FECs are not throwing out good data. These plots can be used in conjunction to the Plane Time Bin plots showing a single event in time and space.

  The number of entries in this plot are tabulated in the same way as the hit distribution plot. It is equal to the number of good events multiplied by the number of clusters per event multiplied by the number of hits per cluster. Note that the number of good events listed at the top of this plot is wrong. That bug has now been fixed.

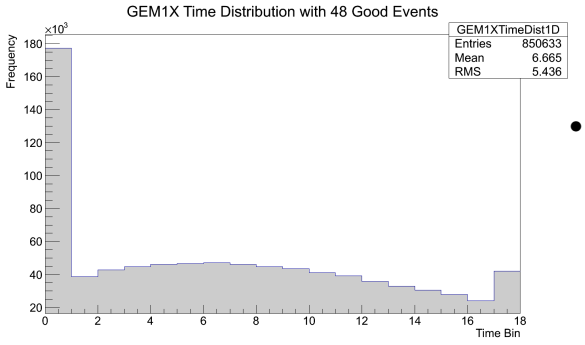

Figure 21: Time Distribution Plot

- Hit Map

  The 2d hit map plots the hits in x strips vs y strips (or equivalently hits in x position vs y position) across the entire detector. Like the previous plots, no track selection occurs with these plots and the number of entries is equivalent to the number of entries in the x hit distribution plot multiplied by the number of entries in the y hit distribution plot. Keep in mind, this implies that many of the hits represent false positives, since we assume that one event has exactly one cluster in each dimension (and no track or cluster selection has been made). To account for this, the top and bottom hit and charge maps exclude all events with a cluster multiplicity in x and y not equal to one. To make the stats more comparable, the side detector plots are not set with this with restriction. As such, symmetric-looking artifacts or false positive events show up on the side hit and charge maps.

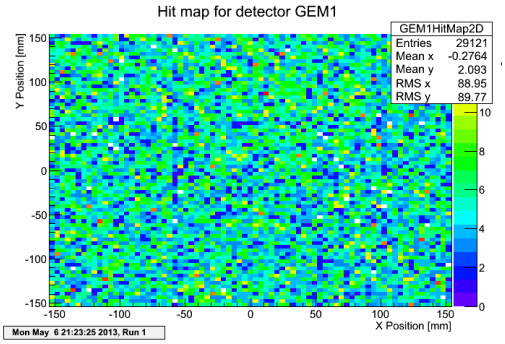

Figure 22: 2d Hit Map

- Charge Map

  The 2d charge map is similar to the hit map in plotting x vs y location but this time it is the charge that is aggregated, not the number of hits. The spacers that are visible in the hit distribution plots are again visible in a checker board type of pattern. Charge across the spacers should be low while charge across the active strips should be uniform. Non-uniformity is a sign of detector bias. Note that the plot shown in Figure 23 is typically shown as the average charge in x and y across all hits in each bin. The current version of our code, however, does not take the average. The fact that the

cosmics provide approximate spatial uniformity allows us to view these plots as a sort of average. If the charge maps are to be used with a station with a radiation source, the average should be taken explicitly.

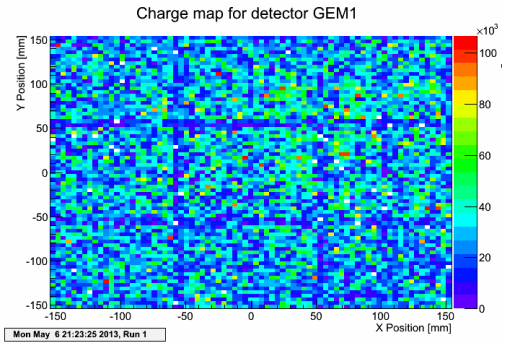

Figure 23: 2d Charge Map

- Charge Sharing

  2d charge sharing plots map the charge sum in the x strip clusters vs the charge sum in the y strip clusters. Since one set of strips physically overlays the other set of strips in the detector, we expect clusters in one dimension to be preferenced slightly more than those in the other. This is visible in the x dimension in Figure 24. However, this preference is only slight, and in general, we expect a symmetric relationship between clusters of one dimension vs the other.

  There is also a correlation between cluster charge sum and cluster size and our ability to localize a particular event. The higher the charge sum, presumably the more strips have been activated and the lower our resolution. However, a high ADC count could also be indiciative of a more energetic event.

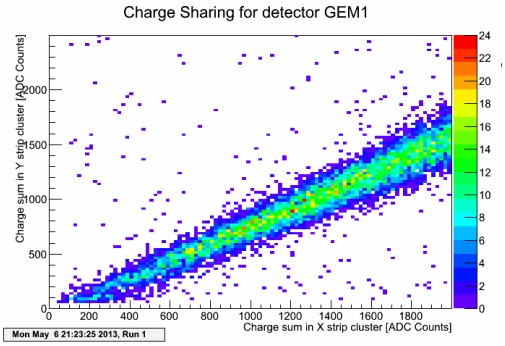

Figure 24: 2d Charge Sharing

- Event Display

  Whereas the previous plots are all monitoring histograms, the event display was designed to extrapolate the entire 2d path of incoming and outgoing tracks. This provides a final big picture look at detector monitoring, and over large enough samples all active strips should register hits. It also shows any geometrically preferenced portions of the detectors.

  It is important to note that unlike the monitoring histograms that do not use track selection, the event display requires it. It uses the AMORE hit files as input, and unless we are running the analysis portion of AMORE, we cannot use the event display. To save time, all analysis is currently done offline, and thus the event display cannot be used during online monitoring.

  In the display, hits are marked by white dots, incoming tracks are red and outgoing tracks are blue. Four detectors are visible in the XZ projection and all eight in the YZ projection. Since the side detectors are localized in the YZ dimensions, side detector hits show up as randomly scattered dots in the XZ projection.

  The event display can also be configured to show only those events passing through a particular target within the active area.

  This display will eventually be integrated into the amoreGui to provide easy ac-

cess. In the meantime, the source is located within the amoreSRS/src/ui director, and the program can be accessed by typing "amore -d SRS -m SRSEventDisplay" in a terminal. Or on the amorePC, it can be accessed by the shortcut command "SRSEventDisplay". To work properly, an offline analysis run must be started and the agent used should match the agent shown in the event display source. There was a configuration file for the display, but for trouble shooting reasons this is now deprecated. This code needs to be cleaned up in the future but works fine for the time being.

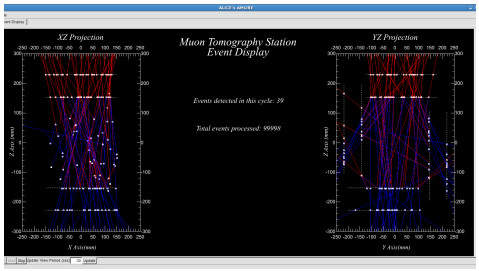

Figure 25: Event Display

VII. Analysis

The analysis methods described below are all done offline. They occur after data is taken and processed and typically require poca files as input. The aggregate program is currently configured in the Eclipse IDE and saved under the name "POCA analysis". All of the plots shown below were generated through the configuration file "PocaConfig.txt" found in the POCA analysis folder. The main function is "PocaAnalysis.cc". Eclipse itself is accessed on the AmorePC by the command "cd eclipse; ./eclipse".

For all these plots, please keep in mind that MTS analysis involves two important and distinct sources of information: scattering angle and POCA statistics. Dense objects have a higher distribution of Coulomb scattering and also generate more POCA points. The analysis techniques described rely on these two ideas.

VII.1 POCA Plots

Currently, 2D POCA plots are the primary visualization during MTS analysis. They use the POCA files generated in AMORE and plot each of the points and accompanying scattering angle in 2D histograms.

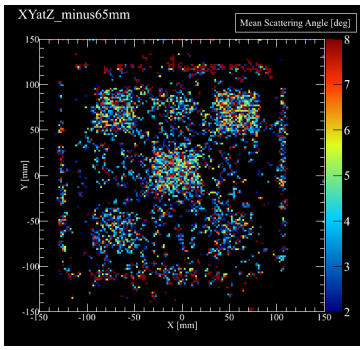

Figure 26: POCA plot from the leadbox3x fcenario

To construct POCA plots, first make sure that you have added the correct poca text file to the POCA analysis directory. Then open the PocaConfig.txt file and set "nInputFiles:" as "1". Then change the name of "inputFile1" to reflect the poca file you just added. Since POCA plots are only 2d plots, the abscissa refers to the axis that is held constant. If you want to see POCA plots for all three pairs of dimensions, change "abscissa" to "XYZ". The output directory is set to a particular file location and the scenario name will be the name shown on each plot. The world variables are the absolute length of each spatial dimension in the MTS in millimeters. In the current station, we usually approximate this with 300 for each dimension. The center of the station will then be aligned halfway between each length. The depth of your slice refers to the depth of the abscissa axis. For example, if you are making XY POCA plots, a 40 mm slice means that each plot will include 40 mm in the Z dimension. The meaning of the variables "nOffset" and "constant" are rather complicated and explained in the conifiguration file. In short, however, they are defined such that we automatically generate 40 mm slices (for example) at every 5 mm interval along the abscissa. As a general rule,

we keep "nOffset" at 16 and "constant" at 40. The bin size refers to the dimension of each histogram bin in the non-abscissa dimension. This means that the 3d voxel area is equal to $binsize * binsize * slice$. The minValue and maxValue variables set the color scale for the POCA plots.

The next set of values refers to the cuts made to particular plots. Before a point is placed in a bin, it undergoes a min and max angle cut. If the scattering angle at that poca point is less than the configured min angle cut or greater than the max angle cut, that point is excluded. After all the bins are filled a further cut, called "min muon cut", may be made to each bin. If that bin has fewer POCA points than the min muon cut, all entries in the bin are erased and its value set to 0. The final cut is a neighboring muon cut (NMC) and it simply aggregates the contents of each bin's eight neighboring bins. If the number of POCA points in those eight bins is fewer than the NMC, the contents of the central bin are erased and the bin set to 0. Note that the color of each bin is equivalent to the combined scattering angle of all POCA points in that bin.

The histo output section is used to define whether you want to generate POCA plots, stats plots or both. If you do not want to output spatial resolution plots and scattering density plots, make sure "spatialRes", "nScenarios" and "nObjects" are all set to 0.

VII.2 Statistics Plots

Statistics plots (or stats plots) are similar to POCA plots; the only difference being that stats plots ignore the scattering angle. The color value of each bin corresponds to the raw number of POCA points found in the bin. The same cuts applied to POCA plots are applied to stats plots.

The importance of stats plots becomes clear when observing the POCA distribution plots described below. Both scattering angle and POCA statistics are capable of discerning the nuclear makeup of objects of interest, but POCA statistics appear to paint that picture quicker. Whereas scattering angle is highly dependent on the resolution of the detectors and the accuracy of the track reconstruction, the simple presence of a POCA point is indicative of higher levels of scattering and avoids the false conclusions that a precise scattering measurement might imply.

To construct stats plots, follow the same instructions found in the POCA plot section.

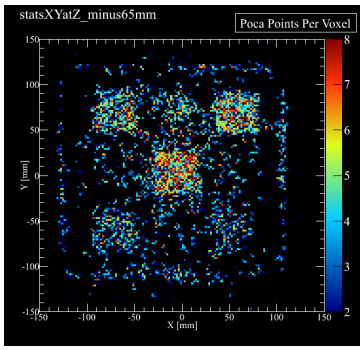

Figure 27: Stats plot from the leadbox3x scenario

VII.3 Spatial Resolution

The horizontal and vertical resolution of our DAQ system is an important metric in proving the viability of muon tomography. Moreso, our use of GEM detectors is largely predicated on the idea that they provide better spatial resolution than drift tubes. One way to visualize this resolution is through spatial resolution plots.

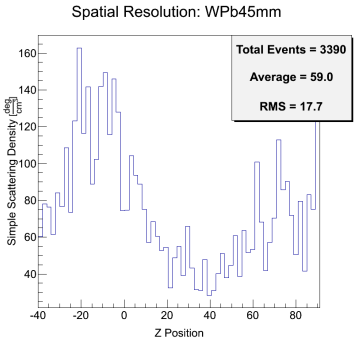

Figure 28: Spatial resolution plot from the WPb45mm scenario

These 1d plots map total scattering as a function of location. We often use them in scenarios in which two objects are placed closely

together in the same plane. If we can resolve the empty space between the two objects, then we can claim a horizontal or vertical resolution at that distance.

To generate these plots without POCA/stats plots, first make sure your number of input files is set to 1 and the name of that file is the same as the poca file you added to the project directory (minus the .txt extension). If you only want to generate spatial resolution plots, make sure the "pocaPlots", "statsPlots", "nScenarios" and "nObjects" variables are set to 0 and the "spatialRes" variable is set to 1. Then enter the 3d min and max coordinates of the volume you wish to analyze. This area should completely encompass the objects of interest and include the empty space between the objects.

VII.4 Scattering Distribution

Scattering distribution plots define explicit volumes of interest within the MTS and bin the scattering angle of each POCA point for each volume defined. The resulting distribution is smoothed with a Landau curve and normalized by dividing each bin by the total number of POCA points within the volume. As a rule, we expect the highest proportion of scattering angles to be low regardless of the density of the object being imaged. However the mean is shifted slightly to the right for heavier objects and the tail should be slightly higher. In the figure shown below, the "No" label represents noise and is defined as any volume inside the MTS that does not include the objects of interest.

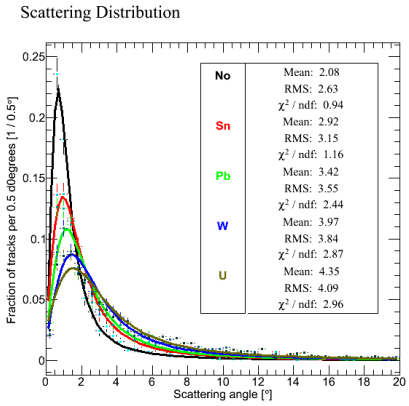

Figure 29: Scattering distribution plot

To generate these plots, make sure you have added your POCA text file to the project folder. Then turn off the POCA, stats and spatial resolution plots by setting "pocaPlots", "statsPlots", and "spatialRes" variables to 0, and activate scattering and poca distribution plots by setting "nScenarios" to the number of scenarios you would like to run at once and "nObjects" to the number of objects you would like to define. If you would like to turn on the "noise" variable mentioned above, change that value to 1. Then, as an example, follow the following format for each object you wish to define:

object1 Pb
minX1 -20
maxX1 23
minY1 -23
maxY1 20
minZ1 -32
maxZ1 11
radLen1 0.5612.

VII.5 POCA Distribution

While scattering distribution tests the ability of scattering angle to differentiate objects, POCA distribution plots test the utility of POCA statistics alone. In general, we expect an $\frac{1}{\sqrt{x}}$ relationship between Coulomb scattering and radiation length. So by plotting normalized values

of POCA points in each object vs the radiation length of that object, we can test the ability of the MTS to discriminate these objects. By comparing POCA and scattering distribution plots at low statistics, we have found that POCA statistics are a better indicator of object density at low statistics than scattering angle. This has implications in fast detection algorithms.

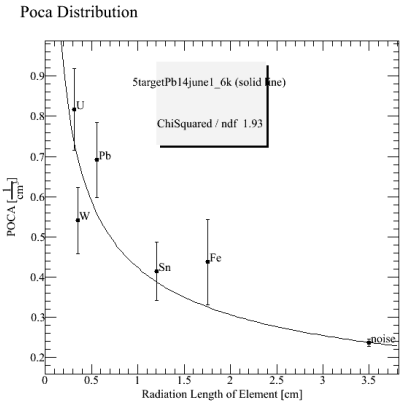

Figure 30: POCA distribution plot

To generate these plots, follow the same directions discussed above under scattering distribution. POCA and scattering distribution plots will be generated at the same time.