2019

Monitoring Software and Charged Particle Identification for the CLAS12 Detector

William A. Oliver
Virginia Commonwealth University

Monitoring Software and Charged Particle Identification for the CLAS12 Detector

A thesis submitted in partial fulfillment of the requirements for the degree of Masters.

By

William A Oliver

B.S. Physics, Virginia Commonwealth University, May 2012

B.S. Mechanical Engineering, Virginia Commonwealth University, May 2012

Virginia Commonwealth University, Department of Physics

Richmond, Virginia

August 2019

TABLE OF CONTENTS

List of Figures

Abstract

The CEBAF Large Acceptance Spectrometer for the 12 GeV era, known as CLAS12, uses the time of flight (TOF) system to identify charged particles from scattering events between the beam and target. The TOF system is divided into two parts: The *Forward* time of flight system, and the *Central* time of flight system. These two sub-systems subtend different polar angles of the detector geometry for wide acceptance of scattered particles.

Reconstruction is the service used to identify particles from the interactions between the beam and target, called as a *vertex* or the point where the interaction occurs. The vertex position is traced back using the tracking system and the TOF system. The resolution of the detector affects the accuracy of the reconstructed vertex location. This paper's goal will be to develop software for validation suite for CLAS12, which will include central and forward tracking plots. Plots will be developed to check the precision of the reconstructed vertices in both the central and forward detectors. This will be done assuming a target with zero dimension at $v_z = 0$, and an extended target of 5 cm at $v_z = 0$. This paper will also look at the TOF resolution, and identify particles using the TOF detectors and the effect of the vertex correction on the velocity vs. momentum plots.

Acknowledgements

I'd like to acknowledge my advisor Dr. Yelena Prok for her guidance and understanding spent during this process. I'd also like to acknowledge Dr. Raffaella De Vita, scientist and developer with the CLAS12 software team. Her work and ideas were the bases for this project.

1. Introduction and Theory

1.1 The Thomas Jefferson National Accelerator Facility

The Thomas Jefferson National Accelerator Facility or 'JLab' for short, formally known as the Continuous Electron Beam Accelerator Facility or CEBAF, is a national lab located in Newport News, Virginia used for conducting electron scattering experiments within the 5-12 GeV energy range. The facility is co-owned and operated by the U.S. Department of Energy, and the Southeastern University Research Associates. JLab was first constructed in the late 1980's and has been conducting experiments since 1995 and features two linac accelerators with three experimental halls. The two linac accelerators work to produce a continuous electron beam with a luminosity of $10^{35} cm^{-2} s^{-1}$.

Figure 1. Aerial View of the CEBAF Facility [18]

Recently, the entire accelerator system has undergone a series of upgrades to accommodate a higher beam energy form the old 6 GeV beam, to a 12 GeV electron beam. The upgrades include a new Central Helium Liquefier to accommodate 10 new cryomodules added to the beam line, a new bending arc, and the addition of experimental hall D located on the opposite end of the accelerator from the other three halls; namely A, B, and C. Each of the three existing experimental hall detectors were also upgraded to accommodate the higher beam energy.

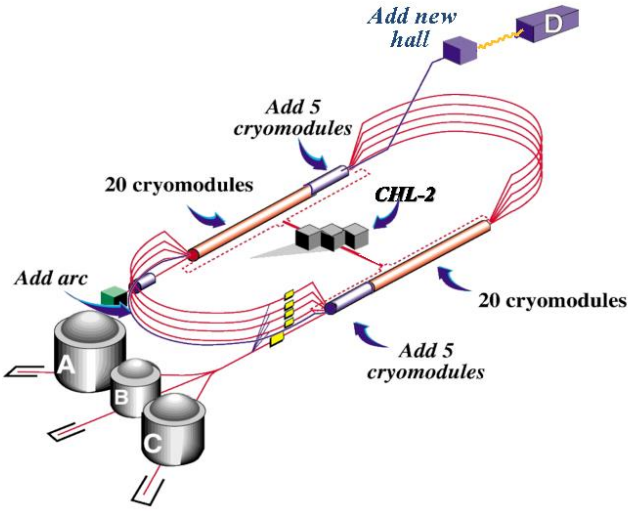
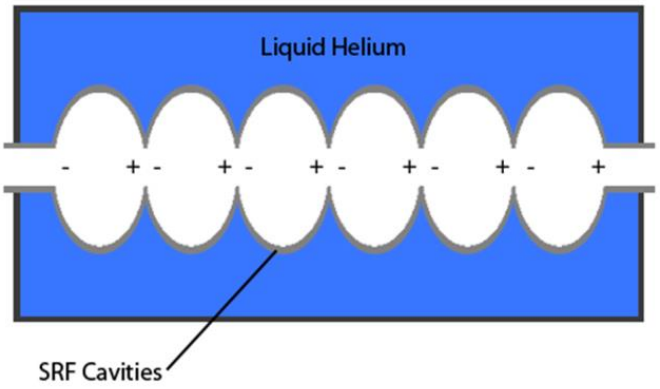


Figure 2. Schematic of the 12 GeV Upgrade [18].

Each hall receives an electron beam from the linac accelerator (except hall D, which uses breaking radiation produced when the beam impinges on a diamond target), wherein the beam encounters a target material often comprised of light elements and their isotopes like hydrogen, helium, or perhaps chemical compounds like ammonia. The electron beam then may interact with the target material to produce scattered particles, which are then analyzed by the detectors. The CEBAF Large acceptance Spectrometer for the 12 GeV upgraded beam energy, or CLAS12, is the experimental detector located in Hall B. It receives beam electrons directly from the linac at 12 GeV, where they impinge on a target (usually hydrogen) to produce scattered particles.

The facility is famous for its full-scale use of Superconducting Radio Frequency Cavities (SRF), which are a special type technology used for accelerating bunches of electrons in a continuous fashion around the accelerator system. The SRF cavities rest in cryomodules that contain liquid superfluid helium -a state of helium at approximately 2 degrees above absolute zero- to maintain the superconducting state of the SRF cavities which are made from pure niobium, and to efficiently conduct heat away from the SRF cavities during operation. Electron bunches are propelled through the cavities as each of the nodes within the cavity change polarity, in a similar fashion to that of a magnetic levitation train system. The term “Radio Frequency” in SRF comes from fact that the electrostatic oscillations in the cavities are in the radio frequency regime.



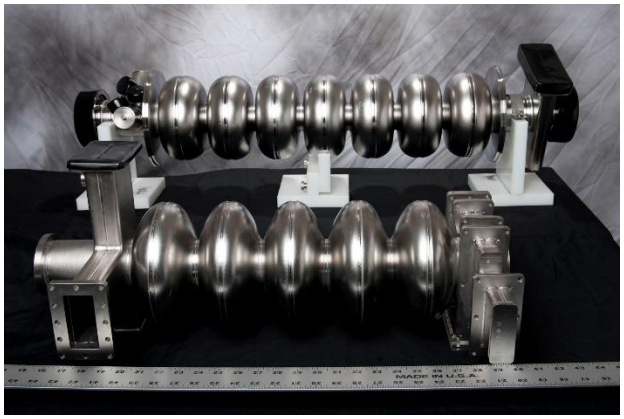


Figure 3. Schematic View of SRF cavity (top) and picture of SRF Cavity [19].

1.2 Theoretical Background

The CLAS12 experiment is designed to further investigate the theory of quark hardon duality, absolving the structure of the nucleon at higher energies in the deep inelastic scattering (DIS) regime at higher Q^2, and with more detail than the valence quarks, or into regions of the sea quarks and gluons.

Scattering experiments with electron beams on hydrogen targets performed at SLAC and CERN in the 1960's revealed that the proton had a structure unlike that of a single particle. The scattering data revealed that the sub-structure was composed of at least three particles with no extended spatial structure, as was confirmed by later experiments. Mathematically this is explained by taking the limit as $k^2 \to \infty, \nu \to \infty$, leading to the structure functions becoming finite functions of a single variable known as Bjorkian momentum x:

$$\frac{\nu}{M} W_2(k^2, \nu) \rightarrow F_2(x)$$

$$2W_1(k^2, \nu) \rightarrow F_1(x)$$

This is known as Bjorken scaling in DIS. The experiments showed that the sub-structure of the nucleon was comprised of three point like fermions with fractional charge and spin 1/2, and constituent electrically neutral massless vector particles with spin 1. These constituent parts, or 'partons', would later be named 'quarks', and the boson particle would later be known as the 'gluon'.

Paradoxically the Baryons seemed to violate the Pauli exclusion principle, i.e. the rule that states that no two fermions may occupy the state at the same time. The three quarks inside proton and neutron seemed to violate this, since these three fermions with spin ½ or -½ all occupied the same state at once. This led to the idea that quarks and gluons carry an extra characteristic described as *color charge*. Since three quarks occupy the nucleon, each parton must exhibit one of three properties; Red, Green, or Blue and that the sum of these three color charges must add to a colorless particle. This is known as *color conservation*. In addition, anti-quarks also must show this characteristic, but in the converse as in case of regular quarks, namely anti-Red, anti-Green, and anti-Blue. This helps explain why Mesons are colorless, being composed of a quark anti-quark pair, each of whose color is the converse of the other, namely blue, and anti-blue, or red and anti-red. Quarks also exhibit other characteristics such as flavors and strangeness as well. Gluons, on the other hand, are the mediators of the strong force, and carry color as well. They are responsible for mediating color charge between quarks. The colors are carried by the eight gluons. These quarks, along with the electron, make up all matter in the universe.

Only colorless particles can be observed in nature, and therefore one may never observe a quark by itself. Instead the strong interaction is described by a force which increases with distance, like a spring, and so the act of pulling quark systems apart results in a quark or anti-quark being pulled from the vacuum. In cases where a baryon is broken apart in deep inelastic scattering experiments, the resulting scattered particles include a Baryon with a new quark replacing the scattered quark (with the same type as the scattered quark), and an anti-quark of the converse color charge accompanying the scattered quark.

Experiments with greater resolution using muons with momenta up to 280 GeV began to uncover more structure within the nucleon, namely the so-called sea quarks comprised of quark

anti-quark pairs, which apart from the three valence quarks, carry a smaller and smaller fraction of initial quark momentum.

1.3 Deep Inelastic Scattering at CLAS12

High energy particle accelerator experiments are used as a way for observing hadrons in detail. At low energies elastic scattering occurs between the target and beam particle, and in this paradigm, the target particle can be resolved. Measuring particle momentum from scattering events gives information about the processes which place during the event. Figure 4 represents an example of such electron scattering experiments. An incident lepton, in this case an electron, with momentum energy E and four momentum $k = (E, \vec{k})$ is incident on a nucleon N. The collision event between the electron and nucleon exchanges a virtual photon with a momentum transfer often expressed as Q^2 (a measure of the virtual photon's inverse wavelength), and the electron scatters at an angle θ with a final four momenta $k' = (E, \vec{k}')$. Note that Q^2 is proportional to the momentum transferred to the nucleus [24].

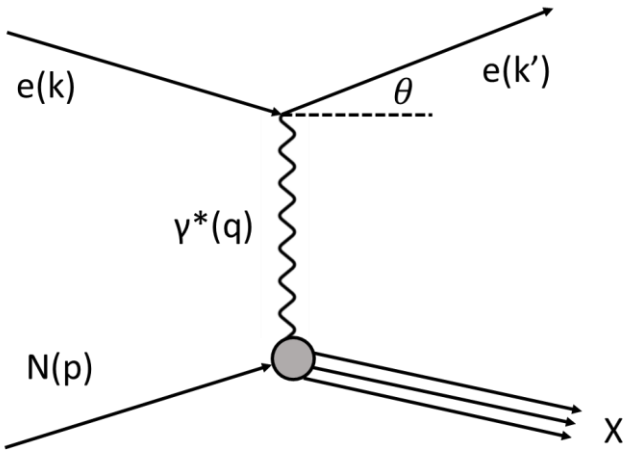


Figure 4. A Feynman Diagram representing the electron-nucleon scattering process. Here, an electron (e) is scattered by a hadron N at an angle θ, exchanging a photon in the process (the wavy line).

Scattering at sufficiently high energies will resolve the constituent partons of the target due to the wave-like nature of the particles whose probability distributions can be described using the uncertainty principle, i.e. the resolution scales with energy. The process where particle collisions interact with the partons with enough energy is known as Deep Inelastic Scattering (DIS). Here,

'Deep' refers to the high momentum transferred to the nucleon, and 'Inelastic' refers to the fact that the nucleon target is 'destroyed' or 'broken', which releases energy.

Experimental data is gained in the form of the scattered particle's energy, momenta, charge and spin. These relate to the scattering characteristics, or the *scattering cross section*, of the target particle, which results in a way which is formed from the deep inelastic scattering processes. The characteristics of the partons form that form the hadron's structure, and their interactions or 'behaviors' with the beam particles, are known as the observables, or observable functions.

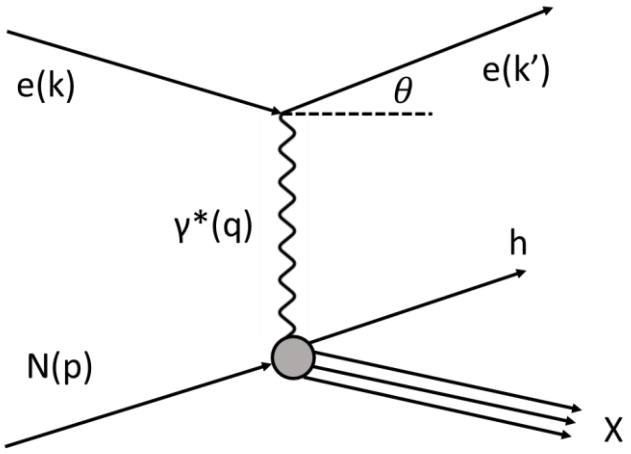


Figure 5. A Feynman Diagram representing the DIS process. Here, an electron (e) is scattered by a hadron N at an angle θ, exchanging a photon in the process, and breaking the nucleon into fragments (X and h).

The CLAS12 detector is designed to measure inclusive, semi-inclusive and exclusive scattering events, ultimately containing clues about the quark transverse momentum and spin correlations. Combined with the energy upgrade, CLAS12 will be able to study single and double spin asymmetries, involving previously unexplored chiral-odd and time-odd distribution functions. [13]

Semi-inclusive deep inelastic scattering (SIDIS) measurements resolve particle scattering from events involving more than the scattered electron, which contrasts with inclusive and exclusive scattering where the former resolves only the scattered electron and the latter resolves all scattered particle from the interaction. In SIDIS the hadron is detected in coincidence with the scattered electron. Deep inelastic scattering refers to the high momentum transfer from the

incident electron to the nucleon. The study quark distributions in the valence quark region is one of CLAS12's main objectives.

In addition to flavor tagging, SIDIS also gives access to the azimuthal final states of the transverse momentum of quarks, and orbital motion of the quarks in the nucleon, which is not accessible in inclusive scattering.

The CLAS12 detector is designed to cover a wide range of kinematics, and in addition to its advance detection capabilities the accelerator provides high beam intensity, high energy, high polarization to allow for the capacity necessary to study the transverse momentum and spin correlations in both the target and fragmentation regions.

Studies can be done on the $\cos 2\phi$ azimuthal asymmetry in unpolarized Drell-Yan processes. The two pion data, π^+ from CLAS12 would allow for the extraction of exclusive two pion asymmetries. Detailed measurements of ALL and its $\cos\phi$ moment as a function of PT in different bins in x,z, and Q^2 combined with measurements of azimuthal moments of the unpolarized cross section proposed for CLAS12 will allow study of the flavor dependence of transverse momentum distributions.

1.4 The CLAS12 Detector

The CLAS12 detector is the detector used in Hall B. It is composed of the series of detector subsystems, which range from the upstream portion of the detector to the downstream portion of the detector. The upstream portion of the detector includes the Silicon Vertex Tracker (SVT), the Central Time of Flight detector (CTOF), the Central Neutron Detector (CND), and High Threshold Cherenkov Counter (HTCC). The downstream portion contains the Drift Chambers (DC), the Low Threshold Cherenkov detector (LTCC), the Forward Time of Flight detector (FTOF), the Forward Tagger, the Electromagnetic Calorimeters (EC), and the PCAL (pre-shower detector)[23]. Together these components are responsible for particle tracking and identification. Solenoid and torus magnets are located in both the upstream and downstream portions of the detector respectively, and are responsible for providing magnetic analysis of charged particles

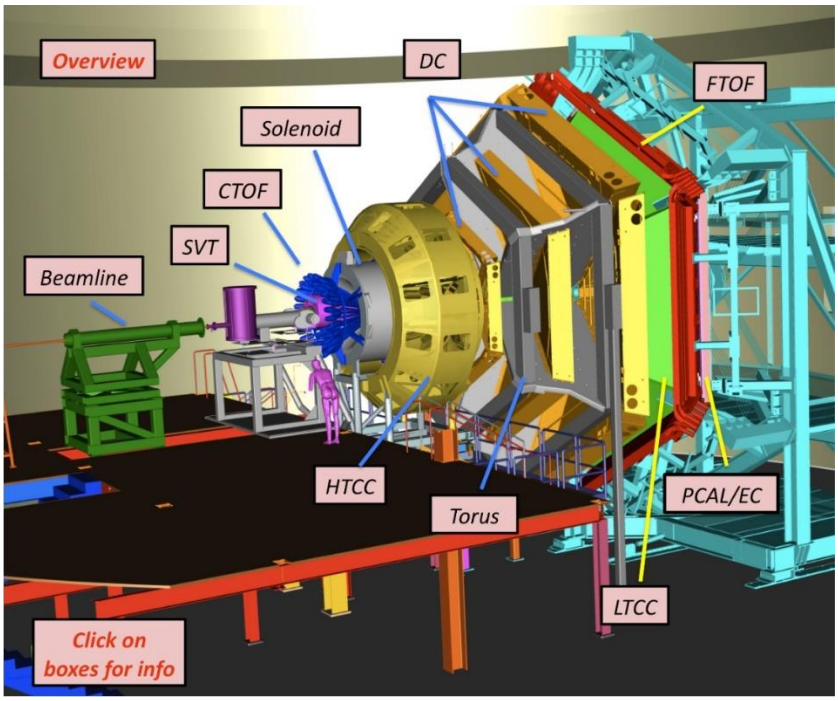


Figure 6. The CLAS12 Detector [23]

Both the SVT and DC are used for tracing particle tracks as they move through the detector, in order to measure the momentum of charged particles emerging from the target. They do so for the upstream and downstream portions of the detector respectively. The HTCC is responsible for generating trigger signals from events from target interactions [20]. The LTCC is responsible for identifying discriminating pions from kaons as part of the forward detector system [22].

2. CLAS 12 Time of Flight System

2.1 Central and Forward Detectors.

The CLAS 12 time of flight System is used to measure the time of flight (TOF) of charged particles emerging from interactions within the target. The TOF system is comprised of two detector sub-systems. The Central Time-of-Flight (CTOF) system, which measures scattered particles at an angular range from $\theta = 35^0$ to $\theta = 125^0$. Specifications for time resolutions call for $\sigma = 60$ ps. It is part of the Central Tracking systems which identifies charged and neutral particles with momenta < 1.5 GeV/C . The CTOF is designed to resolve to resolve pions, kaons and protons with a resolution of 3.3 σ (see Table 1 below). and surrounds the experimental target with an array of scintillation panels. Specifications for the resolution of particles include the separation of particles ranging [10]:

Table 1. CTOF Design Resolution for Particle Identification

CTOF	Design Resolution
π /K separation	3.3 σ up to 0.64 GeV
K/p separation	3.3 σ up to 1.0 GeV
π /p separation	3.3 σ up to 1.25 GeV

The second sub system, the Forward Time of Flight (FTOF) system, is made up of three panels, divided into six sectors, mounted towards the downstream end of the CLAS12 detector array. The forward carriage is roughly 10 m across and is designed to measure charged particles with an angular coverage of $\theta = 5^0$ to $\theta = 45^0$. Each panel is comprised of multiple scintillation bars, with separate panels designed with different time resolutions in mind. Design parameters call for the separation of pions from kaons up to 3 GeV. This means TOF resolutions must be no more than 80 ps at the more forward angles and at least 150 ps at the larger angles [11].

Table 2. CTOF Design Resolution for Particle Identification

FTOF	Design Resolution
π /K separation	4 σ up to 2.8 GeV
K/p separation	4 σ up to 4.8 GeV
π /p separation	4 σ up to 5.4 GeV

The entire TOF system has an azimuthal acceptance of ϕ =+-180^0. Most particles which interact with the target are scattered at low angles through the detector and so this system is important for characterizing the events. Both the FTOF and CTOF system helps resolves particles and they're speed β, which is the particle's speed relative to the speed of light.

Particle TOF is determined by the arrival time of the particle at the TOF counters (after flight path and signals delays are accounted for) and the event start time measured by the tagger system. A more accurate event start time can be determined by referencing the 499 MHz accelerator RF signal, which determines the beam bunch associated with the event, and associating that event with the detected particle. Events can be linked to detected particles within a few ps in this way.

Event reconstruction must also send information like hit information and track fitting for plotting. Reconstruction is split into two parts, to match the configuration of the CLAS12 geometry, which are the *Central Tracking* region, and *Forward Tracking* region. Central tracking using the Silicon Vertex Tracker (SVT) located next to the target and surrounded by the 5T magnetic field. The Forward Tracking region includes the Drift Chambers (DC) and Forward Vertex Tracker (FVT). Vertex fitting algorithms, and track segment fitting algorithms have been developed for each of these tracking regions.

2.2 FTOF Description

The Forward Time of Flight detector is part of the CLAS12 time of flight detector system. The FTOF detector is designed to cover charged particles scattered at polar angles between the range of θ =5^0 to 45^0, with total coverage in the azimuthal range (ϕ =+-180^0). The FTOF detector is mounted on a carriage with its face perpendicular to the beam axis located between the Drift Chambers (DC) and the Electromagnetic Calorimeter (EC). The FTOF detector is divided into 3 panels: panel-1a, panel-1b, and panel-2. Panels are divided into six sections each sub-tending an azimuthal angle of ϕ =60^0. Panel 1a rests just downstream of panel 1b, and both cover angular ranges from θ =5^0 to 35^0. Each of the six sections consists of 23 (panel-1a) and 62 (panel-1b) plastic scintillator bars with double-sided PMT readout sensors. Panel-1a and panel-1b scintillators have a timing resolution of 90 ps to 160 ps and 60 to 110 ps respectively. Panel-2

covers an angular range of θ =35^0 to 45^0, and includes 5 plastic scintillator panels in each section, with a timing resolution of 140 ps to 165 ps [11][6].

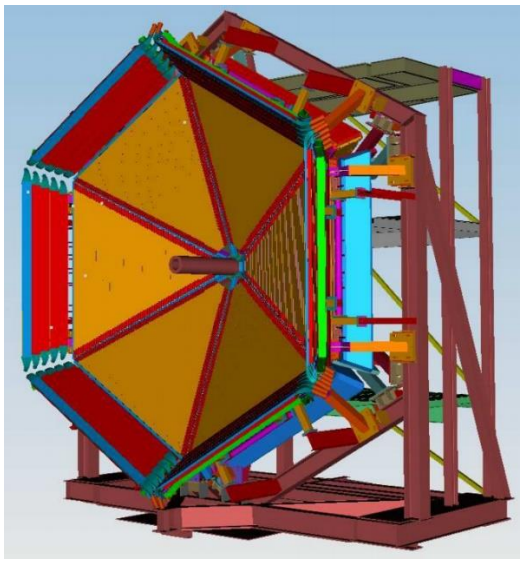


Figure 7. CLAS12 FTOF Detector Array and Platform. Shown in red is the six-sector panel-1b array and shown in orange is the panel-2 array. Panel-1a array is not shown, since this panel rests behind panel-1b. Image credit: D.S. Carman [10]

The scintillator bar lengths in Panel-1a range from 32.3 cm to 376.1 cm, and have a rectangular cross section of 15 cm x 5 cm. Panel-1b scintillator bar dimensions range from 17.3 cm to 407.9 cm in length, with a square cross section of 6 cm x 6 cm. Panel-2 scintillator bars measure 371.3 cm to 462.2 cm in length, with a rectangular cross section of 22 cm x 5 cm.

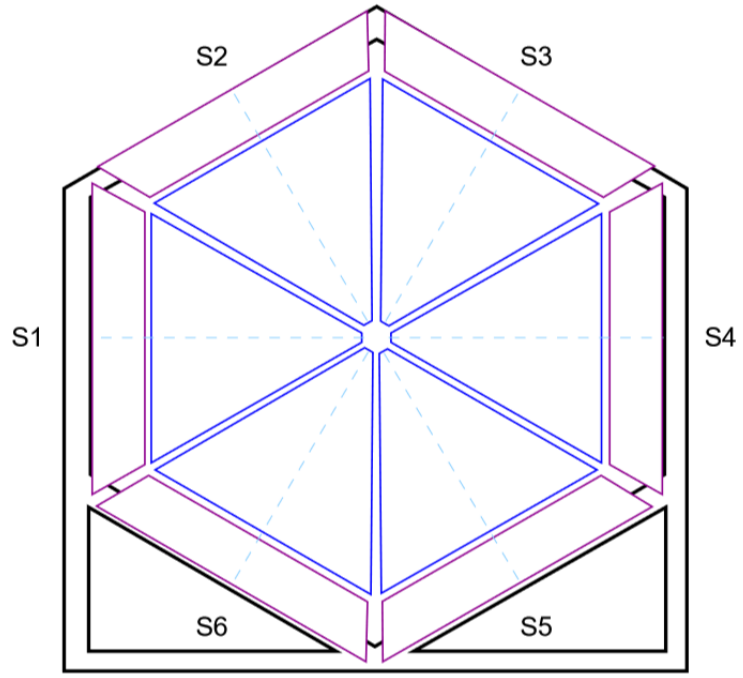


Figure 8. CLAS12 FTOF Detector Panel Array. Image Credit: D.S. Carman [6]

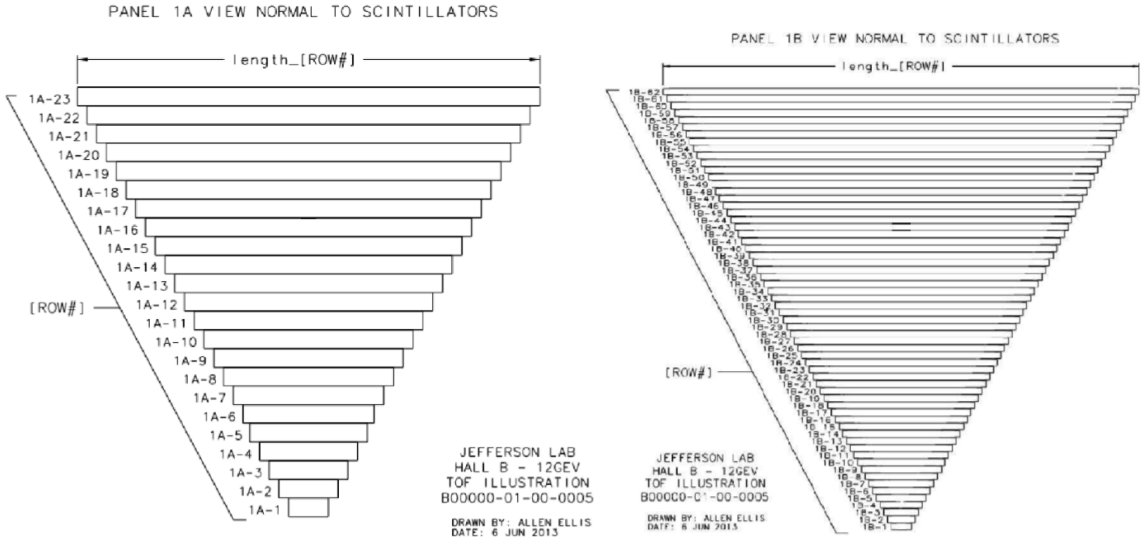


Figure 9. Left: Panel-1a scintillator array, Right: Panel-1b scintillator array. Image Credit: D.S. Carman [6]

Timing resolution specifications call for the separation of charged particles up to 4 σ which translates to 320 ps (see Table 2). Note that the outer scintillator panels have lower timing resolutions, since the resolutions decrease as the length of the bars increase. Also note that the

upgraded CLAS12 detector has ca higher resolution than the CLAS6 detector array due to the higher flux of low angle scattered particles incurred by the increased beam energy.

In contrast to the CTOF scintillator bars, the scintillator panels here lack light guides for efficiency. The resolution, given the parameters of the scintillator panel and PMTs is:

$$\sigma_{TOF} = \sqrt{\sigma_0^2 + \frac{\sigma_1^2 + (\sigma_P L/2)^2}{N_{pe}\exp(-L/2\lambda)}}$$

Here, L is the scintillator length is, and λ is the attenuation length of the scintillator. σ_0 and σ_1 are the intrinsic resolution of the electronics and the jitter within the scintillator and PMT. σ_p is related to the path length variation of the light collected and N_{pe} is the average number of electrons seen by the PMT. Effectively, the σ_{TOF} scales linearly with the length of the scintillator panel, so for this reason the timing resolution is poorer the further away from the center of the array the panel is located [8].

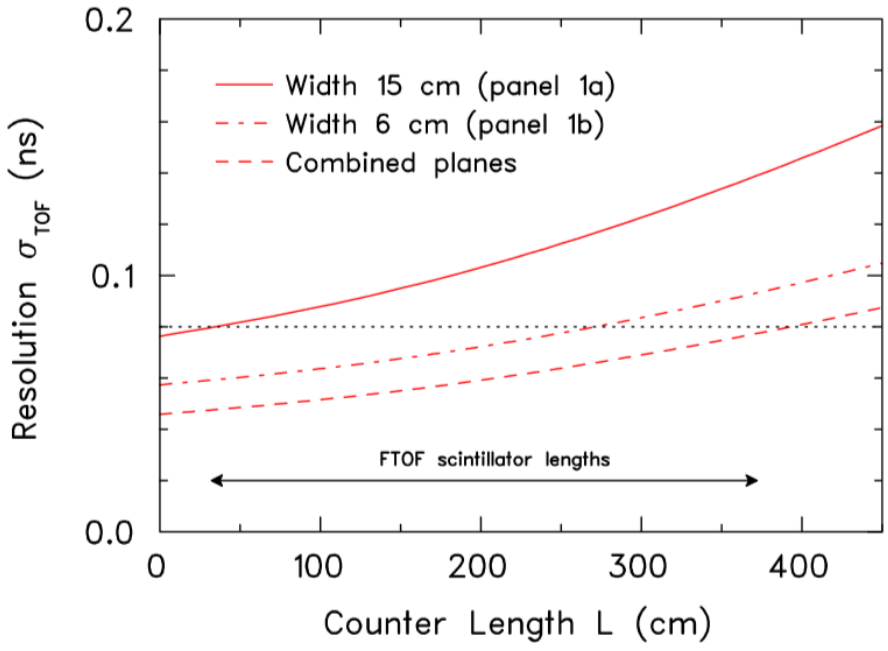


Figure 10. Counter Length vs. resolution. Image Credit: Master Document.

2.3 CTOF Description

The CTOF (Central Time of Flight) Detector system is used to measure the flight time of charged particles from interactions of the beam with the target with an angular acceptance in the range of θ =35^0 to 125^0 and azimuthal acceptance of ϕ =+-180^0. The detector array rests inside a 5 T magnetic field produced by a superconducting solenoid magnet with the main field component in the ϕ direction. The detector array consists of 48, 92 cm long scintillation bars measuring 92 cm long with a 3 x 3.5 cm trapezoidal cross section arranged around the beam axis. The target rests in the center at a radial distance of 25 cm. Each counter has an average time resolution of σ_{TOF}=60 ps along the full length of the detector. Light signals from hits are channeled from each end, focused by light guides and received using two Hamamatsu R2083 PMTs (photomultiplier tubes). Each hit has an associated ADC and TDC value which is used to carry information about the particles' energy and timing information. Timing information is reference from the accelerator's stable radio frequency signal. The two PMT's, both referred to here as the upstream and downstream PMTs, receive signals from particle hits along the scintillation bars of length [5].

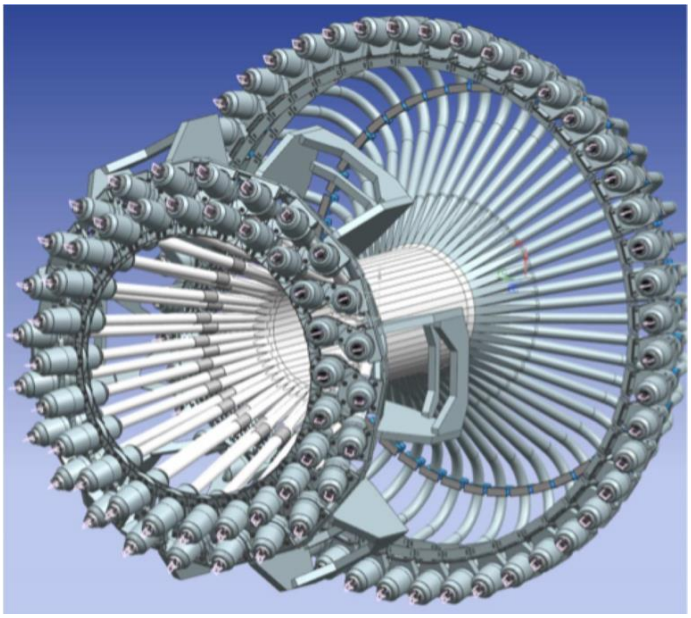


Figure 11. The CTOF Detector. Image credit [5]

One of the central goals critical in the CLAS12 upgrade is to allow for the separation of pions from kaons up to 0.64 GeV and pions from protons up to 1.25 GeV, which calls for at least 200 ps of separation, or a time of flight resolution of $\sigma_{TOF} = 50\ ps$. Time of flight resolution is

determined by the resolution of the PMTs, $\sigma_{TOF} = .5\,\sigma_{PMT}$, so the resolution for the PMTs was aimed to $\sigma_{PMT} \leq 72$ [3]. The following procedure was published in the CLAS12 master document [12].

The combined resolution of the scintillator and light guide, with the efficiency of the two PMTs can be expressed as:

$$\sigma_{TOF} = \sqrt{\frac{(A\sigma_A)^2 + (B\sigma_B)^2}{(A+B)^2}}$$

Where σ_A and σ_B are the timing resolution of each PMT, A and B are the light pulse heights through the straight and bent ends of the tubes respectively. Both A and B are proportional to the transmission efficiency of the light going through either ends of the tubes LTE_A and LTE_B, so if we define f=A/B, and $\sigma_{PMT} = \sigma_A = \frac{1}{\sqrt{f}}\sigma_B$, then σ_{TOF} can be re-written as:

$$\sigma_{TOF} = \sigma_{PMT}\sqrt{\frac{f}{1+f}}$$

Since A and B are proportional to the transmission efficiency of the light going through either ends of the tubes LTE_A and LTE_B, the ratio f is:

$$f = \frac{A}{B} = \frac{\mathrm{LTE}_A}{\mathrm{LTE}_B},$$

and for an asymmetric counter with 1-m long light guides and a PMT with a 77.9 ps resolution:

$$f = \frac{A}{B} = \frac{\mathrm{LTE}_A}{\mathrm{LTE}_B} = \frac{44.6}{35.50} = 1.256$$

$$\sigma_{TOF} = \sigma_{PMT}\sqrt{\frac{f}{1+f}} = 0.746 \times 77.9\,ps = 58.1\,ps$$

Which is less than the efficiency of 50 ps sought after. Later developments lead to an increased time of flight resolution, mostly by increasing the amount of internally reflected light inside the acrylic light guides. This was done by ensuring the cleanliness of the scintillator bars were maintained, and reflective covering was applied to the scintillator bars with an airgap to increase total internal reflection. Resolution is still being improved, but as of now, the CTOF scintillator bars have set a world record for resolution of detector of this kind.

2.4 Scintillator ADC and TDC Measurement Methods

The two PMT's, both referred to here as the upstream and downstream PMTs, receive signals from particle hits along the scintillation bars of length C_L, whose distance from the upstream PMT d_U and distance from the downstream PMT d_D is defined by half the bar length and coordinate *corr* defined with respect to the center of the counter is:

$$d_U = \frac{C_L}{2} + corr\,, \qquad d_D = \frac{C_L}{2} - corr$$

Subsequently, hit times can be measured with respect to upstream and downstream PMTs:

$$t_{hit}^{U} = t_U - \frac{d_U}{v_{eff}}\,, \qquad t_{hit}^{D} = t_D - \frac{d_D}{v_{eff}}$$

Here v_{eff} is the is effective velocity of the light pulse through the detector channel medium, and is set to 16 cm/ns before any further calibration. The average hit time as measured between both PMTs is then:

$$\begin{aligned}\bar{t}_{hit} &= \frac{1}{2}\left(t_{hit}^{U} + t_{hit}^{D}\right)\\ &= \frac{1}{2}\left[t_U - \left(\frac{C_L}{2} + corr\right)\frac{1}{v_{eff}} + t_D - \left(\frac{C_L}{2} - corr\right)\frac{1}{v_{eff}}\right]\\ &= \frac{1}{2}\left[t_U + t_D - \frac{C_L}{v_{eff}}\right]\end{aligned}$$

The hit coordinate, *corr*, is defined by:

$$corr = \frac{v_{eff}}{2}\left(t_U + t_D - upstream\_downstream\right)$$

Where *upstream_downstream* is the timing alignment parameter.

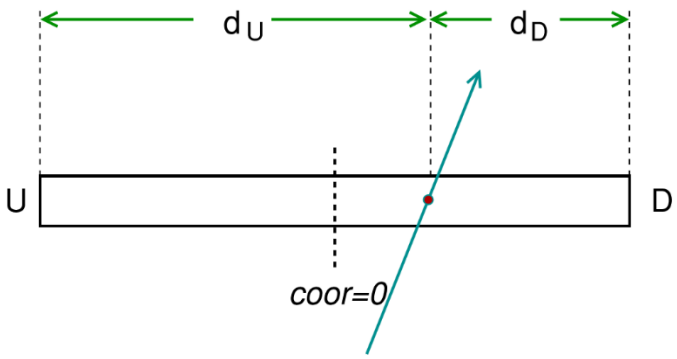


Figure 12. A Scintillator Bar Schematic [5].

ADC values are determined by the deposited energy of the charged particle during its trajectory through the scintillation bar. The effect of muons through each scintillation bar was used to calibrate the rate of energy dissipation, which was found to be dE/dx = 1.956 MeV/cm for minimum ionizing particles in EJ-200.

The energy measured by upstream and downstream PMTs is attenuated by along the length of the bar:

$$E_U = E_{dep} exp\left(\frac{-y_U}{\lambda_U}\right)$$

$$E_D = E_{dep} exp\left(\frac{-y_D}{\lambda_D}\right)$$

Here y_U and y_D, are the distances from the hit location to the PMTs, and E_{dep} is the energy deposited by the charged particle. The geometric mean of the deposited energy between the two PMTs is:

$$\langle E_{dep} \rangle = \sqrt{E_U E_D} = E_{dep}\left[exp\left(\frac{-y_U}{\lambda_U}\right) exp\left(\frac{-y_D}{\lambda_D}\right)\right]^{\frac{1}{2}} = E_{dep} G$$

Where the gain is given by:

$$G = \left[exp\left(\frac{-y_U}{\lambda_U}\right) exp\left(\frac{-y_D}{\lambda_D}\right)\right]^{\frac{1}{2}}$$

A parameterization of the number of photoelectrons measured by each PMT based on studies carried out with the FTOF detectors is given by:

$$N_{pe}^U = E_U \cdot CONV \cdot QE$$

$$N_{pe}^D = E_D \cdot CONV \cdot QE$$

Where,

$CONV = 1800\, photoelectrons/Mev$ deposited and,

$QE = 27\%$, is the quantum efficiency.

Next, a Poisson distribution smear is applied to the computed values N_{pe}^U and N_{pe}^D. This results in an energy smear:

$$E_U^{SMR} = \frac{N_{pe}^{U,SMR}}{CONV \cdot QE}$$

$$E_D^{SMR} = \frac{N_{pe}^{D,SMR}}{CONV \cdot QE}$$

The measured ADC values for both upstream and downstream PMTs are therefore:

$$ADC_U = \frac{E_U^{SMR}}{K} \cdot \frac{1}{G_U} = \frac{E_U^{SMR}}{K}\left[exp\left(\frac{-y_U}{\lambda_U}\right) exp\left(\frac{-y_D}{\lambda_D}\right)\right]^{-\frac{1}{2}}$$

$$ADC_D = \frac{E_D^{SMR}}{K} \cdot \frac{1}{G_D} = \frac{E_D^{SMR}}{K}\left[exp\left(\frac{-y_U}{\lambda_U}\right) exp\left(\frac{-y_D}{\lambda_D}\right)\right]^{-\frac{1}{2}}$$

$$K = \left[\frac{\left(\frac{dE}{dx}\right)_{MIP} \cdot t}{ADC_{MIP}}\right]$$

Where,

$t = \ scintillation\ bar\ thickness.$

Similarly, the timing information for both the upstream and downstream PMTs is approximated using the actual hit times and the time it takes for the signal to propagate from the hit location each PMT:

$$t_U = t_{hit} + \frac{y_U}{v_{eff}^U}$$

$$t_D = t_{hit} + \frac{y_D}{v_{eff}^D}$$

Again where y_U and y_D, are the distances from the hit location to the PMTs, and $v_{eff}^U, and\ v_{eff}^D$ is the effective velocity of light through the counter, which are approximately the same, v_{eff}. Similarly as the ADC signal, a smear is applied to the calculated times t_U, and t_D. A gaussian smear is applied with a standard deviation $\sigma_{PMT} = \sqrt{2}\sigma_{counter}$, where $\sigma_{counter}$ is the resolution of the counter for both upstream and downstream counters. The digitized timing values are the smeared time divided by the TDC channel to time conversion factor, $C_{TDC} = 0.024\ ns/bin$:

$$TDC_U = \frac{t_U^{SMR}}{C_{TDC}}$$

$$TDC_D = \frac{t_D^{SMR}}{C_{TDC}}$$

2.5 Charged Particle Tracking

2.5.1 Event Reconstruction

Event reconstruction is the program which provides track parameters and identify particles on an event basis, whether the data is either real or from Monte Carlo simulation, for physics analysis. Event reconstruction includes the identification of charged and neutral particles using the tracking information from the either the EC or CC, and the TOF (time of flight) information from the TOF system to identify particles. The particle's three momenta can be determined using the particle's track through the EC or CC while the TOF determines the particle's velocity β. The momentum and the velocity are used to determine the particle's mass, *and this is used to identify or distinguish the particle*:

$$m = \frac{p}{\beta\gamma}$$

Neutral particle tracks are identified using the SC or the calorimeters (or both), and the TOF is primarily determined by the time signal in the calorimeters. Charged particle TOF is determined by the CTOF or FTOF detectors. This report will only focus on charged particles [15].

2.5.2 General Tracking

Charged particle tracking must contend with the background noise and the inhomogeneity of the magnetic field found in the detector. Tracks must be identified from the trails left by the particles from the events within the target. This makes the development a procedure to identify tracks quite difficult, and so the efforts of many scientists are involved in developing a particle tracking procedure [15].

Time-based tracking involves the time associated with the tracks through the drift chambers. In general, Before the hit-based tracking (HBT) or time-based tracking (TBT) can be done, three steps are performed:

1. Cluster finding

   Contiguous groups of hits or "clusters" are found in each superlayer. The superlayers are groups of layers within the EC or CC.

2. Track segment finding

A Look-up table is used to find the clusters that are consistent with a track.

3. Segment linking

   Segments from superlayers are linked together. Again, look-up tables are used here.

4. Track fitting

   Linked segments are used to determine a preliminary angle and momentum. A trial track is constructed through the magnetic field of the detector, and the DOCA is calculated with respect to the position determined by the drift times (this is the time-based tracking, or "TBT"). The best fit is chosen by varying the various track parameters estimated earlier. The trajectory of the particle is determined by using five track parameters (see *Kalman Filter* algorithm).

2.5.3 Kalman Filter algorithm

Charged particle tracking using the SOCRAT package was developed for charged particle reconstruction. The key to the reconstruction algorithm is the so called *Kalman Filter* algorithm used in final track fitting, which I have given a derivation of, which is shown in the CLAS12 master document [12].

If a system can be described by an n-vector $\vec{x}_k$ at some discrete step k of its evolution The $\vec{m}_k$ vector here is related to the $\vec{x}_k$ vector through a unitary transformation matrix, which in this case is the measurement matrix:

$$\vec{m}_k = \mathbf{H_k} \cdot \vec{x}_k$$

The state of the system when extrapolated to the next step (k+1), which is the next measurement in practice, can be defined as:

$$\vec{x}_k^{k+1} = \mathbf{F_k} \cdot \vec{x}_k$$

Here $\boldsymbol{F_k}$ is a known propagation operator. Using an extrapolated vector, i.e. a vector of all previous measurements, and the current measurement, an update of the system is performed. Using a covariance matrix of $\vec{x}_k$, $\boldsymbol{C_k}$, and defining the covariance matrix of the extrapolated state $\vec{x}_k^{k+1}$:

$$\mathbf{C_k^{k+1}} = \mathbf{F_k} \cdot \mathbf{C_k} \cdot \mathbf{F_k}^T + \mathbf{Q_k}$$

Here, the matrix $\boldsymbol{Q_k}$ is a noise matrix. Next the χ^2 value can be expressed as:

$$\chi^2 = (\vec{x}_{k+1} - \vec{x}_k^{k+1})^T \cdot \left(\mathbf{C_k^{k+1}}\right)^{-1} \cdot (\vec{x}_{k+1} - \vec{x}_k^{k+1}) + (\mathbf{H_{k+1}} \cdot \vec{x}_{k+1} - \vec{m}_{k+1})^T \cdot (\mathbf{M_{k+1}})^{-1} \cdot (\mathbf{H_{k+1}} \cdot \vec{x}_{k+1} - \vec{m}_{k+1})$$

This is minimized to give the updated n-vector at k+1:

$$\vec{x}_{k+1} = \mathbf{C_{k+1}} \cdot \left(\left(\mathbf{C_k^{k+1}}\right)^{-1} \cdot \vec{x}_k^{k+1} + \mathbf{H_{k+1}}^T \cdot (\mathbf{M_{k+1}})^{-1} \cdot \mathbf{H_{k+1}} \cdot \mathbf{H_{k+1}}^T \cdot \vec{m}_{k+1}\right)$$

The updated covariance matrix is:

$$\mathbf{C_{k+1}} = \left(\left(\mathbf{C_k^{k+1}}\right)^{-1} + \mathbf{H_{k+1}}^T \cdot (\mathbf{M_{k+1}})^{-1} \cdot \mathbf{H_{k+1}}\right)^{-1}$$

In the case of experiment, the n-vector is the 5-vector, which due to symmetry is:

$$\vec{x} = (z, \phi, p_x, p_y, p_z)^T$$

Here, z is the coordinate along the z-axis, ϕ is the coordinate along the azimuthal direction, and p_x, p_y, p_z are the three-momenta. The evolution is with respect to time, t:

$$\begin{pmatrix} z \\ \phi \\ p_x \\ p_y \\ p_z \end{pmatrix}_{t+dt} = \begin{pmatrix} 1 & 0 & 0 & 0 & \frac{dt}{m} \\ 0 & 1 & \frac{-ydt}{mr^2} & \frac{xdt}{mr^2} & 0 \\ 0 & 0 & 1 & \frac{qBdt}{m} & 0 \\ 0 & 0 & \frac{-qBdt}{m} & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} z \\ \phi \\ p_x \\ p_y \\ p_z \end{pmatrix}_t$$

Here r, is the radial distance from the beam axis. Note that the 5x5 middle matrix is the propagation matrix $\boldsymbol{F_k}$. The measuring matrix can be calculated with the transformation matrix:

$$\mathbf{H_k} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{pmatrix}$$

Since the double layer measures z and ϕ, at a given r. The multiple scattering effect can be taken into account by adding the process noise matrix $\boldsymbol{Q_k}$.

2.6 Charged Particle Tracking in The Forward Detector

The drift chamber is divided into three chambers each of which is divided into six sectors subtending 60 degrees each. Track candidates are found using the following [12]:

1. Combinations of clusters (all the hits in a series) with possible track candidates
2. Find a "road" in each of the corresponding clusters
3. Run the Kalman Filter algorithm on tracks where roads were identified

Cluster matching is applied, and a *track segment* is formed with the criteria: $|W_1 - W_2| < 0.18 \times W1 + 5$, where W_n is the wire number associated with the superlayer where the hit was recorded. Track segments are then compared, and a track candidate is defined. Below are two plots illustrating the difference between filtered and unfiltered distributions of hits. The bottom plot shows the final track candidate.

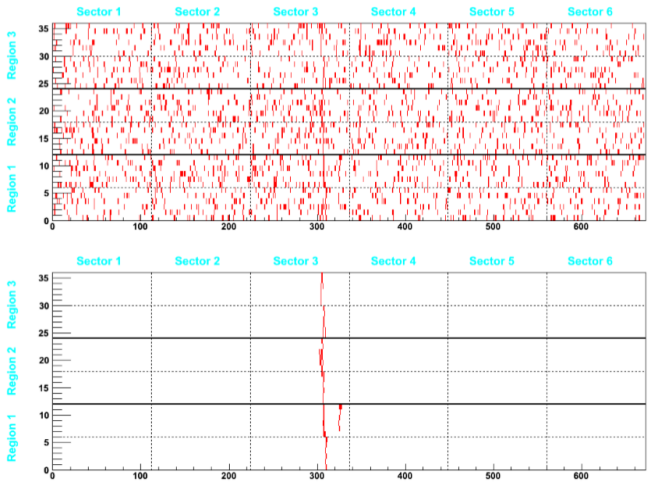


Figure 13. Track Filtering before and after. Image Credit: CLAS12 Master Document [12].

More filtering techniques such as an improved road finding algorithm (currently in development) are used to further discriminate noise from signal, and to solve the so called "left-right ambiguity problem".

The hits measurements are sent to the Kalman Filter with a state vector:

$$\vec{x} = (x, y, u_x, u_y, Q/p)^T$$

Since the geometry of the Forward detector means each measurement has approximately the same z, this coordinate system is chosen. X, and y are coordinates in the xy-plane perpendicular

to the beam axis, u_x, u_y, are slopes of the projected tracks in the xz, and yz planes, and Q/p is the charge divided by the total momentum. This last parameter is calculated using the angles before the track enters region one of the detectors and after region 3 of the detector:

$$Q/p = \frac{\theta_3 - \theta_1}{0.3 \times \int Bdl}$$

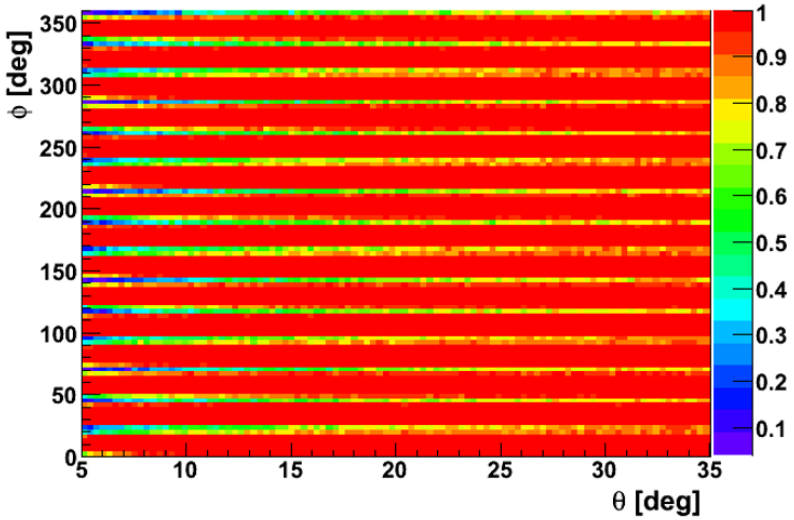


Figure 14. Resolution of the CVT with respect to angle Image Credit: CLAS12 Master Document.

Tracks are then matched with those in the forward tracker. The forward tracker fitting algorithm is very similar to the one used for the SVT in the central detector. Tracking efficiencies with respect to the azimuthal and polar coordinates are shown in the plot above. These show similar dead zones to the ones in the SVT, due the shape of the detector.

2.7 Charged Particle Tracking in The Central Detector

Track finding and track fitting in the Central Tracker involves using selection criteria for filtering the noise (such as background radiation) from the signal (the real tracks from the events). [12]

1. Find hit candidates in each double layer of BST, and find combinations that define a track
2. The strips are polygons (not curved) so the fitting procedure is more complicated.
3. The algorithm finds the hit location by association with the strips
4. The hit candidates are found and filters them based on:

I. The hit candidates should have a track close to a circle
II. The first two hits should be close to ϕ: $|\phi_2 - \phi_1| < 0.7$ or $> (2\pi - 0.7)$
III. The transverse momentum of the first three hits should be within the range $p_T > 0.2$ GeV and < 4 GeV
IV. These hits should also be within: $|\phi_3 - \phi_2| < 0.7$ or $> (2\pi - 0.7)$
V. The circle drawn by the hits should be less than 8 mm from the beam axis
VI. The last hit on the 4th double layer should be close to this fitted circle

This finds 3 or 4 hits to form a track candidate. Track parameters like the transverse momentum p_T are then obtained assuming the track is a helix. The trajectory of the particle is also determined by using five track parameters Kalman Filter algorithm.

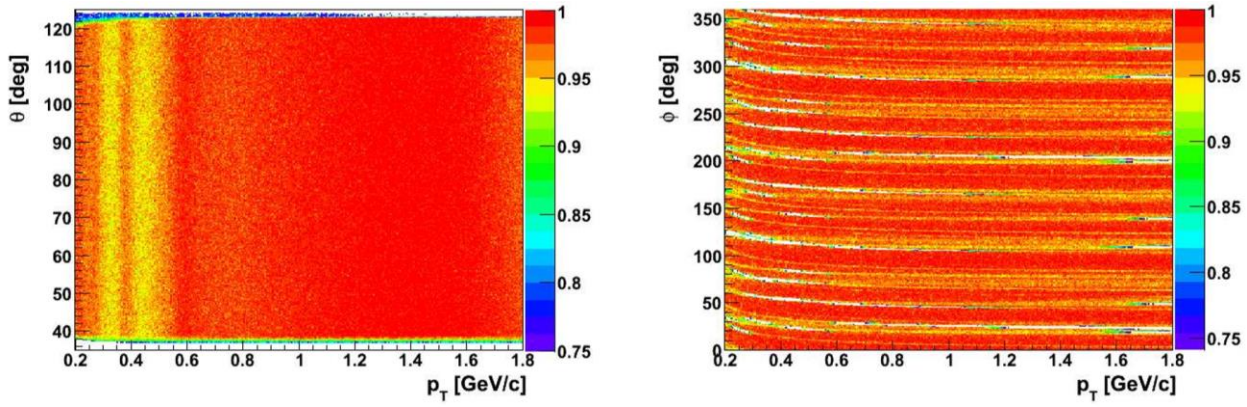


Figure 15. The tracking efficiency as a function of θ (left), φ (right, and transverse momentum p_T. The inefficiencies in the φ direction are due to particle crossing the corners of strips in the detector geometry. Image Credit: CLAS12 Master Document [12].

2.8 Notes about cuts

Accelerator hit data is normally mixed with noise, such as incident electrons from the accelerator beam, scattered electrons that only have minor interactions with the target atoms, and cosmic rays. In addition, interactions with the target don't always occur a manner that interpretable by physicists, and so a simple cut-based approach to selecting events and particle identification can used to filter out unusable data. Cutting can refer to selecting events above a certain momentum or type, separating particles based on charge and performing fiducial cuts across detector planes.

3. Software

3.1 CLAS12 Analysis and Reconstruction Framework (CLARA)

Clara, or the CLAS12 Analysis and Reconstruction Framework, is a distributed systems software framework similar to other popular software applications such as Apache Spark, Storm or Hadoop. These applications exist in the management layer of a system architecture and offer the ability to store and distribute data and processes to massive computing clusters or networks.

CLARA orchestrates the services in order to solve one problem collaboratively. Clara offers real-time data stream processing and Service oriented architecture (SOA) and is specifically developed for physics-based data Processing (PDP) that has a distinct component-based structure. The CLARA framework allows independent, loosely based specialized software components to cooperate to achieve processing goals in either a parallel or series pattern, which is ideal for solving the specific problems presented by the massive streams of data constantly being produced by accelerator detector such as the CLAS12 detector.

The CLARA architecture consists of three layers: The PDP service bus, the Service layer, the PCEP layer and the Orchestration layer [14]. The PDP service bus is layered on top of distributed pub-sub middleware, designed to integrate applications without the need for programming. This layer is built beneath the Service layer which holds services like the ones used for track reconstruction like ClusterFinder, or RegionalSegmentFinder. This service layer provides input to the PCEP layer, which in the case of reconstruction, provides filtration and particle identification. These layers are controlled by the Orchestration layer which provides data flow control, load balancing, and error recovery.

All the different detector component reconstruction services are assembled into a working unit. Here, the various reconstruction services are the green circles. These include the: stream builder (R), stream writer (W), electromagnetic-calorimeter reconstruction (EC), hit based tracking (HT), time of flight reconstruction (TOF), time-based tracking (TT), particle identification (PID), staging (SS), application orchestrator (OR).

One such service that employs the CLARA framework is the Service Oriented Tracking application (SOA). One such service is the Drift chamber track reconstruction composition service, which receives hit information from the from the Drift Chamber detector, and then

groups them into clusters using the ClusterFinder service, and passes them to the RegionalSegmentFinder Service, which links the clusters into region segments. The linked clusters are then passed to the TrackCanidateFinder which associates the clusters into track candidates. CLARA then uses the track candidates from other standalone SOT services such as the Event, EVIOHits, and forward tracker to reconstruct the entire track.

CLARA platform allows for multi-dimensional scaling to utilize all available cores within a node, and across a network of nodes, referred to as vertical and horizontal scaling respectively. Horizontal scaling across a network of nodes, allows one to processes in many threads in parallel, but does not scale linearly due to network latency. Vertical scaling simply allows users to employ more cores on a signal node in parallel, and scales linearly albeit with less processing power than that of a network of nodes. Both scaling techniques are used to varying degrees when processing and depends on the type and difficulty of the process. The massive amount of computing power used over a distributed network allows the user to search massive data sets or iteratively employ regression algorithms a quick manner, whereas a single core may solve simpler linear mathematical models in a much more efficient manner.

The computing resources needed to perform the Monte Carlo simulations, the data acquisition form the detector, and reconstruction which includes the GEMC model and track fitting, and vertex reconstruction algorithms requires large amounts of computing power. Around 13 petabytes of data storage is required for reconstruction, along with approximately 10,000 CPUs or GPUs. Simulation and DAQ require 5 and 2.5 petabytes of data storage respectively, with the simulation requiring around 5700 CPUs or GPUs.

3.2 Notes about Data Flow

Data generated by CODA (CEBAF Online Data Acquisition) is contained in a format called Event IO (EVIO), and is not supported by C++, C or, JAVA. EVIO format has a hierarchical data structure based on the accelerator bank structure. EVIO files are limited in size, and since they cannot be read by the any of the OOP languages used by CLARA, they must be translated into the hipo file format. This is called cooking. Cooking takes the EVIO files and chained them together into a larger file that can be processed and saved at once. These are the data banks, which take the form of columnated data in the form of json files.

3.3 Monte Carlo Simulation

A simulated CLAS12 detector is built from scratch using the Geant4 libraries from provide by CERN. Simulations include the physical interaction of particles within materials, nuclear interactions, multiple scattering, and pair production. The magnetic fields associated with the solenoid and torus magnets are also simulated, along with their particle interactions. The GEMC is modeled as accurately as possible to the specification set by engineering drawings and actual measurements, so the output from the MC simulation can emulate the detector as accurately as possible.

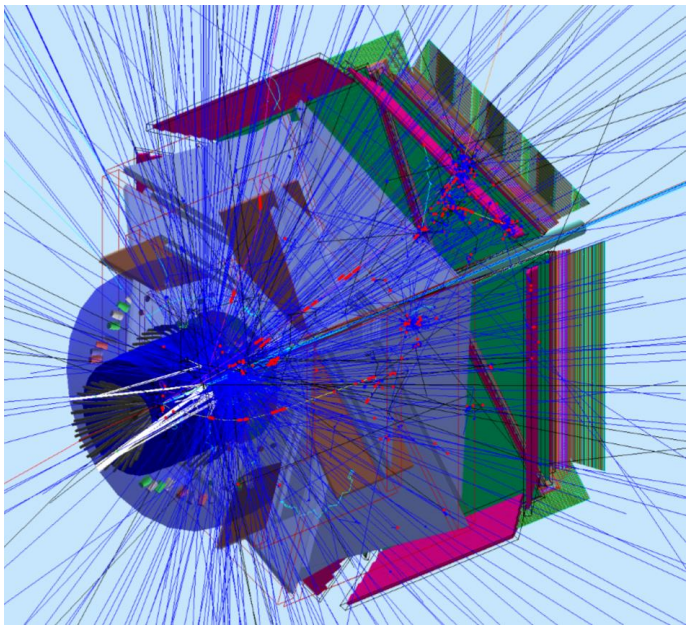


Figure 16. GEMC simulation. Image Credit [2].

Simulated runs start with an *event* generator, which includes a list of input file containing particle four vectors, vertex positions, and particle id's (PIDs). Simulations run with Monte Carlo, and the reconstructed particle parameters are compared to the input particle parameters [2].

3.4 Geant4

Geant4 (Geometry and Tracking) is a platform used for simulating detector geometry, experimental parameters, particle collisions and tracking. Originally developed by CERN in the

late 90's this software is no open source and is used widely by many nuclear/high energy experiments.

All material constants scattering cross sections and magnetic moment distributions are simulated. Detector geometry is modeled using CAD software, which includes the simulated magnetic fields and material properties like cross section absorptions magnetic susceptibility and optical properties. Calibration constants and inefficiencies can be added later as well. Target nuclear cross section is also included in the simulation setup. Using the Monte Carlo simulation Transport calculations, hit definition, detector sensitivity, and outputs such as the Bank definitions are generated. Particle interactions are also simulated using SIDIS Monte Carlo [2].

3.5 Descriptions of the TrackingTest.java package:

Plots were developed in the TrackingTest.java package for the CLAS12 validation suite. The TrackingTest.java package code can be found in the index section. This package was originally developed by Dr. Rafaella Devita of the CLAS12 software development team at Jefferson Lab. New Routines were developed for producing plots in addition to the ones produced by Dr. Devita. The TrackingTest.java package uses data banks in the form of json files to build. The data banks are produced from reconstruction either by using real detector data or simulated data from Monte Carlo. Specifically, TrackingTest.java performs the following:

- Gets data from the banks and performs cuts, like momentum cuts, or other selection criteria such as fiducial cuts.
- Creates histogram plots from the data.
- Fits gaussian curves to the histogram plots.
- Produces simple statistics from histogram plots which include the means, amplitudes, and standard deviations.
- Produce plots and display them.

The TrackingTest.java package is divided into 12 constructors or methods contained within the TrackingTest class. The method processEvent() gets data banks, and performs selection criteria and calculations for producing the plots. The fitting methods, createHistos(), fitVertex(), fitMC(), and fitMCSlice() are used for creating the histograms and fitting gaussian curves to the data. The method printCanvas() uses the java libraries to produce and display the plots.

Below are some examples of the routines developed for the purposes of producing the plots presented herein.

3.5.1 Vertex Comparison

Here is the pseudocode developed for the Vertex comparison plots. An event must have particles detected in the forward and central trackers in order to compare the two reconstructed vertices, so the first if statement provides this check. The code then loops over all particles found in the CVT bank, and in this case, chooses the positive particles. Before looping over all particles in the CVT bank, the code loops over all positive particles detected by the forward detector, and compares the vertices of each of these particles to the one from the CVT. The Code does this for each positive particle found in the CVT. A scatter plot and a histogram are produced.

Algorithm 1 Vertex Comparison

```
1: if event has been detected in forward and central tracker then
2:     loop Over the rows in the CVT json file
3:         if particle in CVT row is positive then
4:             loop Over the rows in the TBT json file
5:                 if particle in TBT row is positive then
6:                     fill scatter plot with vz of the TBT vs. vz of CVT track.
7:                     fill histogram with (vz of TBT track)-(vz of CVT track)
```

3.5.2 Beta vs. Momentum

In this code, the beta is gotten from the value calculated by reconstruction. The bank REC::Particle must have particle produced by reconstruction. The event must have an electron detected in the Forward Detector (FD), and that that electron is the scattered beam electron specified by being the first particle in the REC::Particle bank. Beta is then plotted for the central and forward detectors, for both positive and negative particles.

Algorithm 2 Beta vs. Momentum

```
1: if event has been produced by reconstruction then
2:     if Event has e associated with it, and e has been detected in FD then
3:         loop Over all other particles in the event
4:             if particle is detected in forward detector then
5:                 if particle is positive then
6:                     plot momentum of particle vs. velocity.
7:                     fill histogram with velocity of each particle)
8:                 if particle is negative then
9:                     plot momentum of particle vs. velocity.
10:                    fill histogram with velocity of each particle)
11:            if particle is detected in central detector then
12:                if particle is positive then
13:                    plot momentum of particle vs. velocity.
14:                    fill histogram with velocity of each particle)
15:                if particle is negative then
16:                    plot momentum of particle vs. velocity.
17:                    fill histogram with velocity of each particle
```

3.5.3 Beta vs. Momentum Using Beta Calculated from Scratch

Events are selected in which the first particle is an electron detected in the forward detector.

The electron FTOF time is gotten by looping over the REC::Scintillator bank and selecting the entry which was detected in panel-1b and with the variable pindex=0 (this means the particle detected in the scintillator is the electron associated with the event). The hit time and path of the electron are gotten here. The even start time is calculated as:

$$t_{\,event\ start\ time} = t_{hit,e} - \frac{L_e}{c}$$

Next, the particles are chosen from the banks, and the electron event start time and vertex corrected time is used to calculate the new beta:

1. $v_{\,z,corr} = \frac{v_{\,z,bank}}{c}\ or,\ v_{\,z,corr} = 0$
2. $\Delta t = t_{\,event\ start\ time} + t_{RF} + t_{rf,const}$

3. $t_{corr,rf} = MOD(\Delta t, t_{rf,const}) + \frac{t_{rf,const}}{2}$

4. $t_{corr} = t_{\ event\ start\ time} + t_{corr,rf}$

5. $\beta = \frac{L_i}{t_{hit,i} - t_{corr}} \frac{1}{c}$

Algorithm 3 Beta vs. Momentum from Scratch

```
1: if event has been produced by reconstruction then
2:     if Event has e associated with it, and e has been detected in FD then
3:         loop Over all particles in scintillator bank
4:             if if electron, and if was detected in panel-1b then
5:                 EventStartTime = hittime-Path/c
6:         loop Over all other particles in scintillator bank
7:             vzcorr=particlevertex/c, or
8:             vzcorr=0
9:             deltat=-EventStartTime+RFTime+RFConst-vzcorr
10:            RFCorr=MOD(deltat,RFConst)-RFConst/2
11:            CorrStartTime=EventStartTime+RFCorr
12:            beta=path/(hittime-CorrStartTime)/c
13:            if particle is positive then
14:                plot momentum of particle vs. beta.
15:                fill histogram with beta of each particle
16:            if particle is negative then
17:                plot momentum of particle vs. beta.
18:                fill histogram with beta of each particle
```

3.5.3 FTOF Vertex Comparison

The FTOF vertex algorithm simply chooses events detected in the forward detector and gets the vertices from the banks for the events. First, this is plotted with the entire forward detector in mind, and histograms are produced. Next, the vertices for each of the six sectors are plotted, and histograms are produced.

Algorithm 4 FTOF Vertex

```
1: if event has been detected in forward tracker then
2:     loop Over all particles in the TBT bank
3:         if particle is positive then
4:             plot vertex histogram.
5:             if particle momentum is greater than 2 Gev/cc then
6:                 plot vertex histogram with momentum cut.
7:         if particle is negative then
8:             plot vertex histogram.
9:             if particle momentum is greater than 2 Gev/c then
10:                plot vertex histogram with momentum cut.
11:        if particle was detected in sector 1 then
12:            if particle is positive then
13:                plot vertex histogram.
14:                if particle momentum is greater than 2 Gev/c then
15:                    plot vertex histogram with momentum cut.
16:            if particle is negative then
17:                plot vertex histogram.
18:                if particle momentum is greater than 2 Gev/c then
19:                    plot vertex histogram with momentum cut.
20:        ....
```

4. Plots and Data

4.1 Forward Detector Vertex Comparison

The forward detector system consists 5 sets of triangular sections mounted just upstream of the central detector. Naturally there exists a misalignment between the determined forward detector and CVT vertex locations due to misalignment in the geometry and instrument timing imprecision. The central vertex tracker is thought to be a few hundred microns misaligned with the target, along with a 50-micron misalignment of the silicon vertex tracker. With respect to the CVT, the forward detector is thought to be misaligned at a level of several mm. By finding the events with at least two particle tracks, where both detectors detect at least one of these tracks from the same vertex, and by measuring the distribution between the differences in vertex locations with respect to each detector, the misalignment between the two detectors can be determined.

A histogram can be used to determine the mean displacement between the vertex locations. A routine that selects semi-inclusive or exclusive events where both Vertex trackers measure the tracks of the vertex location was used. The mean within the gaussian distribution gives the misalignment with respect to the beam direction. Both reconstruction banks measure vertex position with respect to the ideal beam line. In addition to selecting these events, the differences were compared only between the positive and negative particles separately.

Positive and negative tracks were plotted separately. Momentum cuts were placed at p>2 Gev/C for both positive and negative particles. Positive particles whose scattered angle was greater than θ =10 were chosen since most scattered positive particles were in this region (see figure 17). Negative particles whose scattered angle was greater than θ =15 were chosen, since most electrons below this threshold are mostly electrons partially scattered by the electron beam.

Please see figures 18, and 19 for vertex histograms. Positive particle vertex locations for Sectors 3 and 6 show furthest deviation from $Vz = 0$ at 0.321 cm and 0.337 cm respectively, though the vertex displacement is even larger for the negative particle in these same panels at 0.441 cm and 0.491 cm respectively.

Table 3.

Positive Particles			
Sector #	Mean(cm)	σ (cm)	σ/c Time resolution in (ps)
sector 1	0.088	0.824	27.4856814
sector 2	0.219	0.818	27.285543
sector 3	0.321	0.91	30.3543327
sector 4	0.073	0.808	26.9519789
sector 5	0.263	0.829	27.6524635
sector 6	0.337	0.867	28.9200071
Negative Particles			
Sector #	Mean (cm)	σ (cm)	σ/c Time resolution in (ps)
sector 1	0.082	0.561	18.7129457
sector 2	0.084	0.589	19.6469252
sector 3	0.441	0.676	22.5489328
sector 4	0.06	0.543	18.1125304
sector 5	0.12	0.609	20.3140534
sector 6	0.491	0.648	21.6149534

Specification call for a time of flight resolution of σ =80 ps. Assuming the particles travel near the speed of light, the standard deviation of the reconstructed vertex location, we can see the time of flight resolution of the particles in the left column in table _. They range between 18ps-30ps, therefore each panel meets the timing resolution from the CLAS12 specifications.

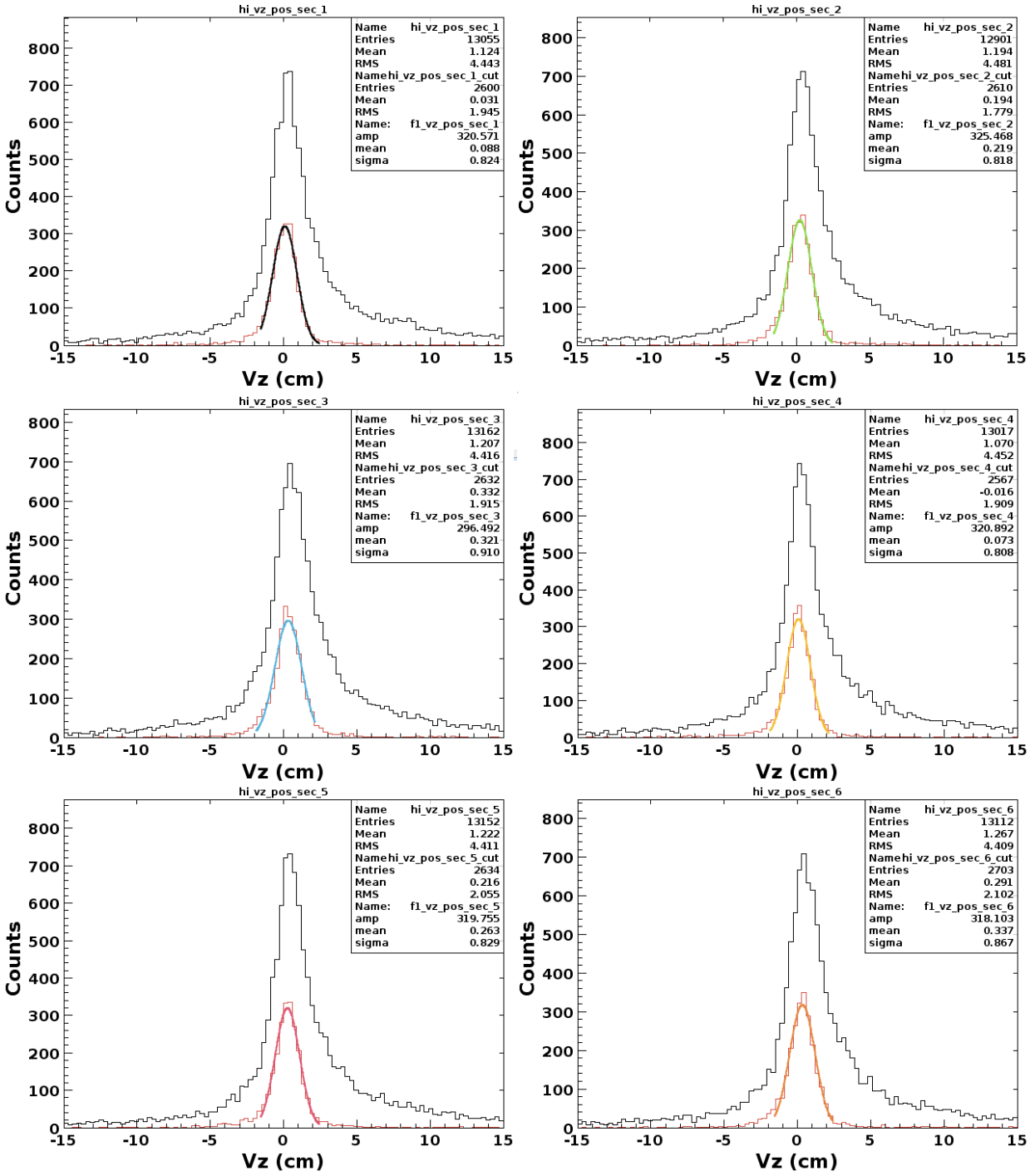


Figure 17. Red distributions have cuts of p>2 and θ >10. The misalignment is with respect the z direction. Panel misalignment and other factors contribute.

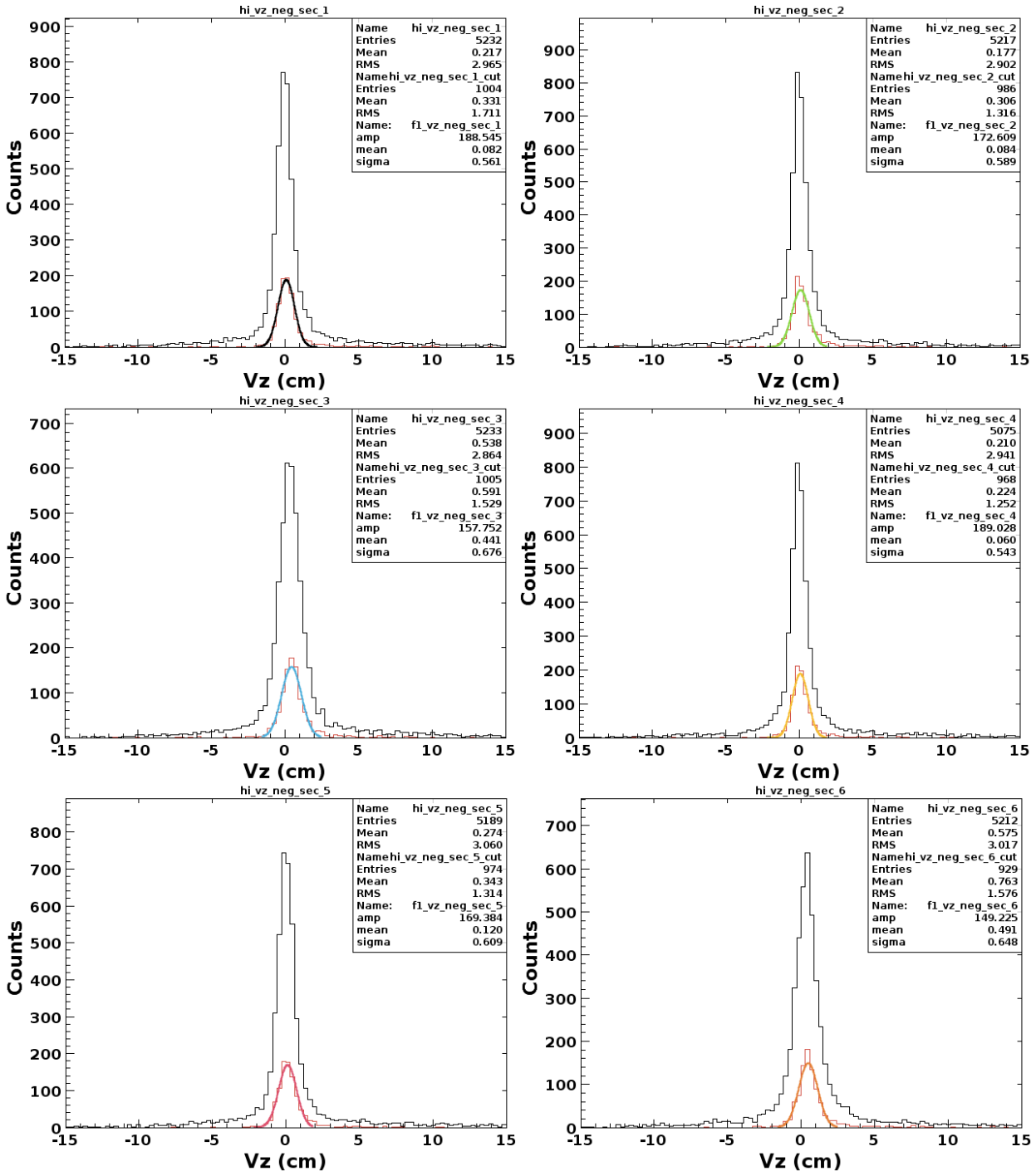


Figure 18. Red distributions have cuts of p>2 and θ >15.

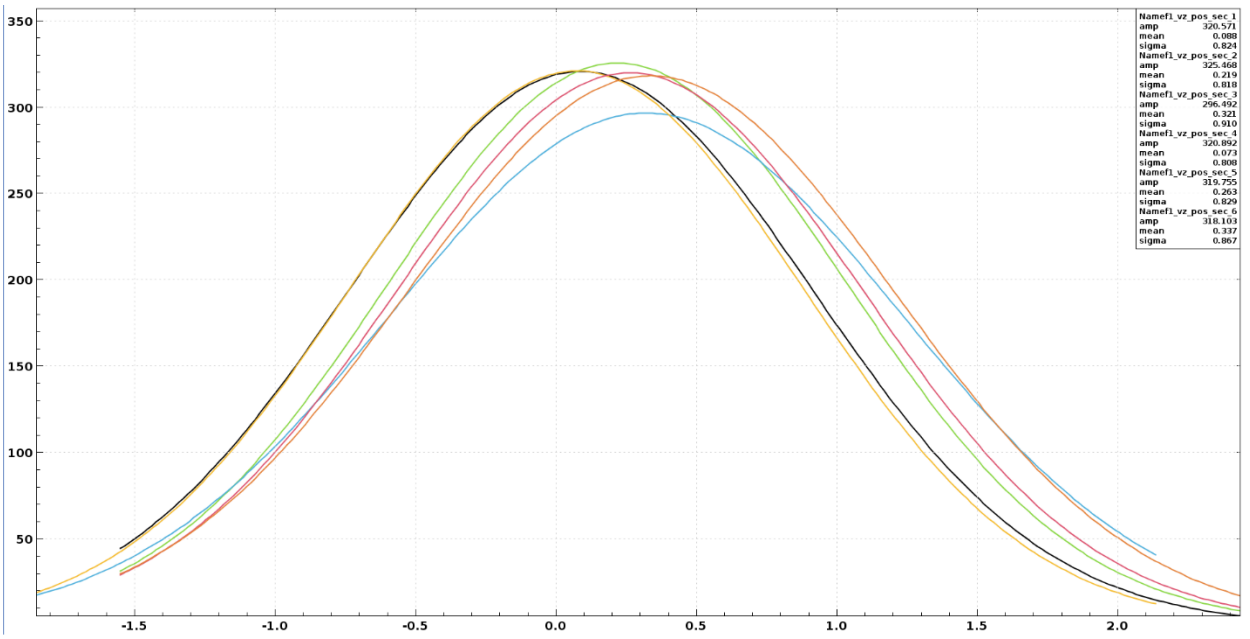


Figure 19. Black-Sector 1, Green-Sector 2, Blue-Sector 3, Yellow-Sector 4, Pink-Sector 5, Orange-Sector 6.

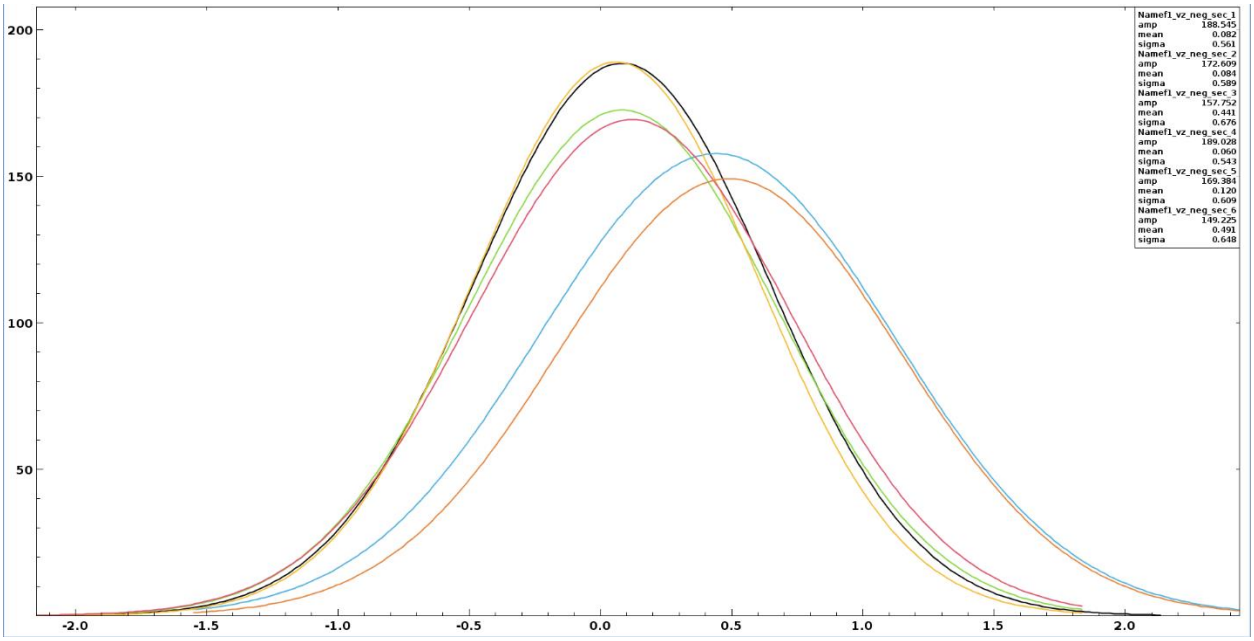


Figure 20. Black-Sector 1, Green-Sector 2, Blue-Sector 3, Yellow-Sector 4, Pink-Sector 5, Orange-Sector 6 Similarly, Sectors 3 and 6 show furthest deviation from Vz=0.

4.2 Forward Vertex Distributions

Vertex reconstruction depends on the time of flight of particles through the detector. Vertex reconstruction is critical for finding particle coincidences from particle interactions with the beam, and so the forward detector is tuned to find tracks from fast moving pions, kaons and electrons from events at the target location. The forward detector subtends an angular range of θ = 5 to 45 degrees and ϕ = ∓180 degrees. Six sectors work to resolve the vertex locations from tracks with low scattering angles.

Angular θ dependence on the vertex location can show timing offsets across the FTOF scintillator panels, often due to anisotropies in magnetic field distributions, detector geometry and even wiring calibrations. Ideally, there should be symmetrical or flat displacements across the θ axis in the scatter plot in figure 21 and (center).

Here we can see most positive particles tend to scatter at low degrees of θ, due to their heavier mass. Positive particle distributions in figure 21 show most particle tend to scatter at θ = 12. Plot shows no bias towards the vertex location with respect to θ.

However, we also see also see a large grouping of negative particles scattering at low angles as well. These are the beam electrons that tend to pass through the target relatively unaffected, so their high velocity and low scattering angle from the target interaction allow them to pass through the magnetic field relatively unaffected. Negative particle distributions in figure 22 show most particles tend to scatter at θ = 10. Plot shows some bias towards the vertex location at higher degrees of θ as shown by the flare towards the lower right quadrant of the graph.

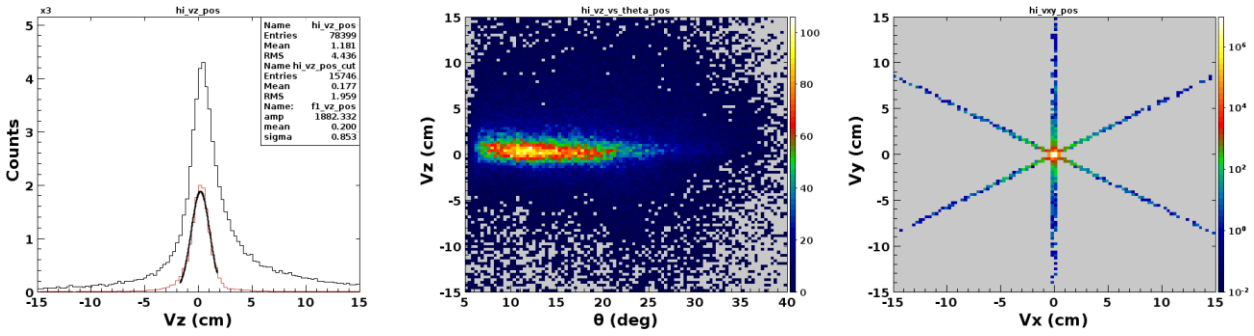


Figure 21. Forward Detector Vertex Positions for Positive Particles.

The forward detector vertex distributions were plotted for positive and negative particles. The distributions for the upper left plot represent the vertex distribution for the positive particles as recorded by the forward detector from a hydrogen target with polarized and unpolarized beam. No cuts were placed on the larger distribution seen in the graph. Mean vertex location for all positive particles was $v_z = 1.181\ cm$. Particles with higher momenta, above 2 GeV/c, have a cleaner path through the detector magnetic field due to their higher energy, and so these particles give more precise measurements. Positive particles with higher degrees of deflection above $\theta = 10$ were also added to this cut. Mean for this distribution was $v_z = 0.177\ cm$, and the fitted gaussian mean was $v_z = 0.200\ cm$.

The distributions for the lower left plot represent the vertex distribution for the negative particles as recorded by the forward detector from a hydrogen target with polarized and unpolarized beam. No cuts were placed on the larger distribution seen in the graph. Mean vertex location for all positive particles was $v_z = 0.332\ cm$. Negative particles with higher degrees of deflection above $\theta = 15$ were also added to this cut, to avoid beam electrons and partially scattered electrons. Mean for this distribution was $v_z = 0.423\ cm$, and the fitted gaussian mean was $v_z = 0.210\ cm$.

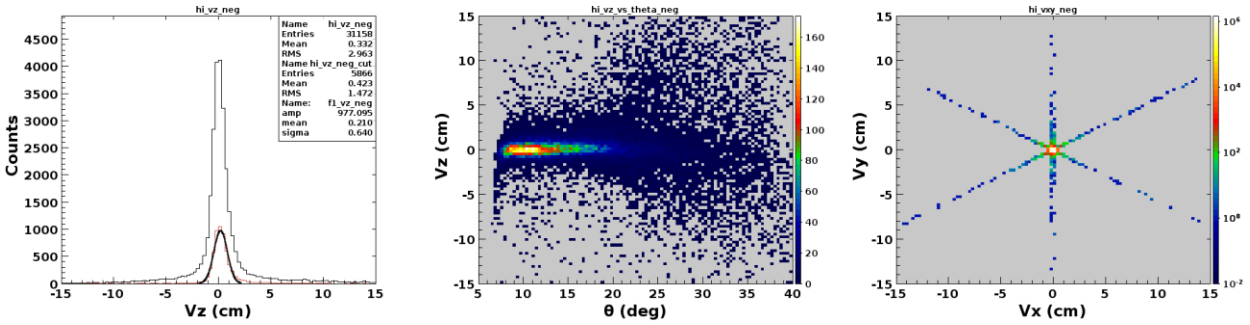


Figure 22. Forward Detector Vertex Positions for Negative Particles.

Right two graphs in figures 22, 23 (right) show the vertex distribution of positive and negative particles using the reconstruction across the six sectors of the forward detector. These distributions were later improved in future version of reconstruction 6b.2.0. Distributions here are seen with respect to the x-y plane.

4.3 Central Detector Vertex Distributions

Vertex reconstruction using the time of flight system also depend on the second detector used for timing. The CTOF and CVT detector subtends an angular range of θ = 45-125 degrees and ϕ = 180 degrees. An array of scintillator panels are used to find tracks which contain electrons, pions, kaons, and protons. The CTOF assigns the hit point as the midpoint between the entrance and exit points along the scintillation panel. Currently, there is an issue with particles moving through the downstream end of the scintillation bars, since this portion of the bars are bent (see figure 23). "This leads to path length errors up to 10% of the overall pathlength depending on the track local angle through the counter and the hit position along the bar" [17].

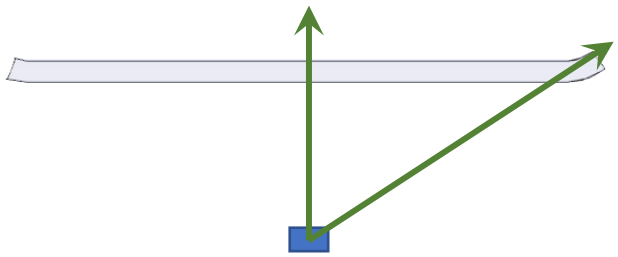


Figure 23. Scintillator Bar Geometry. Image Credit: D.S. Carman [17]

The CVT vertex distributions were plotted for positive and negative particles. These plots represent the vertex distribution for the positive particles as recorded by the CTOF and CVT from a hydrogen target with polarized and unpolarized beam. No cuts were placed on the larger distribution seen in the graph. Mean vertex location for all positive particles was $v_z = 0.819\ cm$. Particles with higher momenta, above 2 GeV/c were included in the smaller distribution in figure 19. Mean for this distribution was $v_z = 0.764\ cm$, and the fitted gaussian mean was $v_z = -0.915\ cm$.

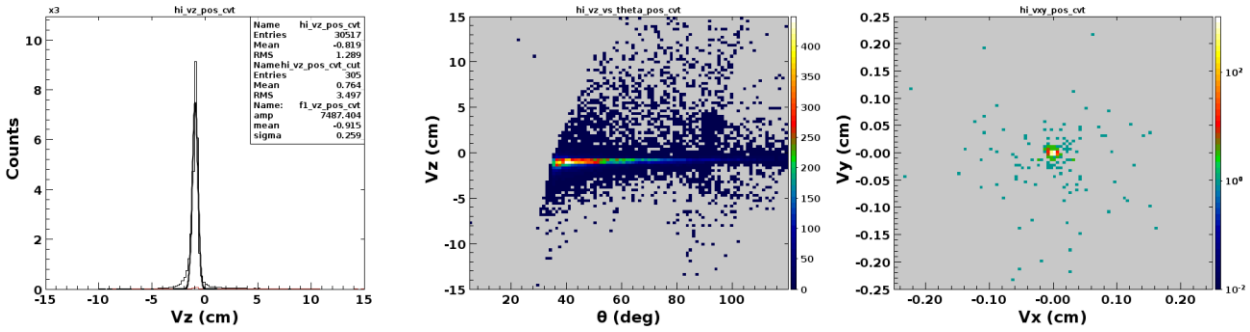


Figure 24. CTOF Vertex Positions for Positive Particles.

The distributions for the lower left plot represent the vertex distribution for the negative particles as recorded by the CVT from the same hydrogen target with polarized and unpolarized beam. No cuts were placed on the larger distribution seen in the graph. Mean vertex location for all positive particles was $v_z = -0.722\ cm$.

The middle two plots in figures 25, and 26 are the reconstructed vertex locations for the CVT detector, for positive (top) and negative (bottom) tracks. Distributions do show some angular anisotropy towards the lower degrees of θ. This is probably an effect due to the downstream portion of the scintillation counters being curved, as mentioned earlier.

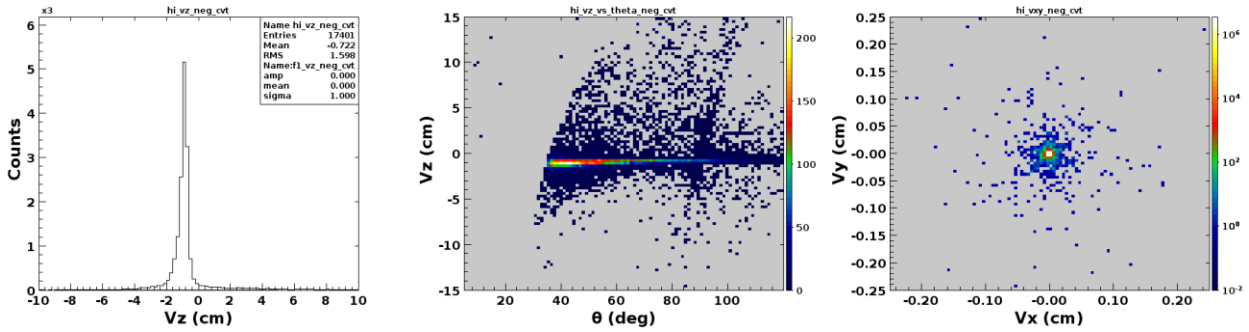


Figure 25. Vertex Positions for Negative Particles with Respect to the CVT.

Right plots in figures 25, and 26 show the vertex distribution of positive and negative particles as seen across the x-y plane.

Positive tracks for the CTOF are shown in figure 26. Scattering angle with respect the particle's momentum shows a similar pattern to the forward detector, indicating particle's with higher momentum scatter at smaller angles. We can see how the reconstructed particles taper off with increasing angle. Vertex reconstruction shows a mean of -0.915 cm, with a standard deviation 0.259 cm. Chi values are quite low with a mean of 4.338. Angular dependence of ϕ shows a pattern which could be due to magnetic field effects on particle path or detector geometry. (Ask about this). Particles should take a helical path through the silicone vertex tracker before reaching the scintillation array.

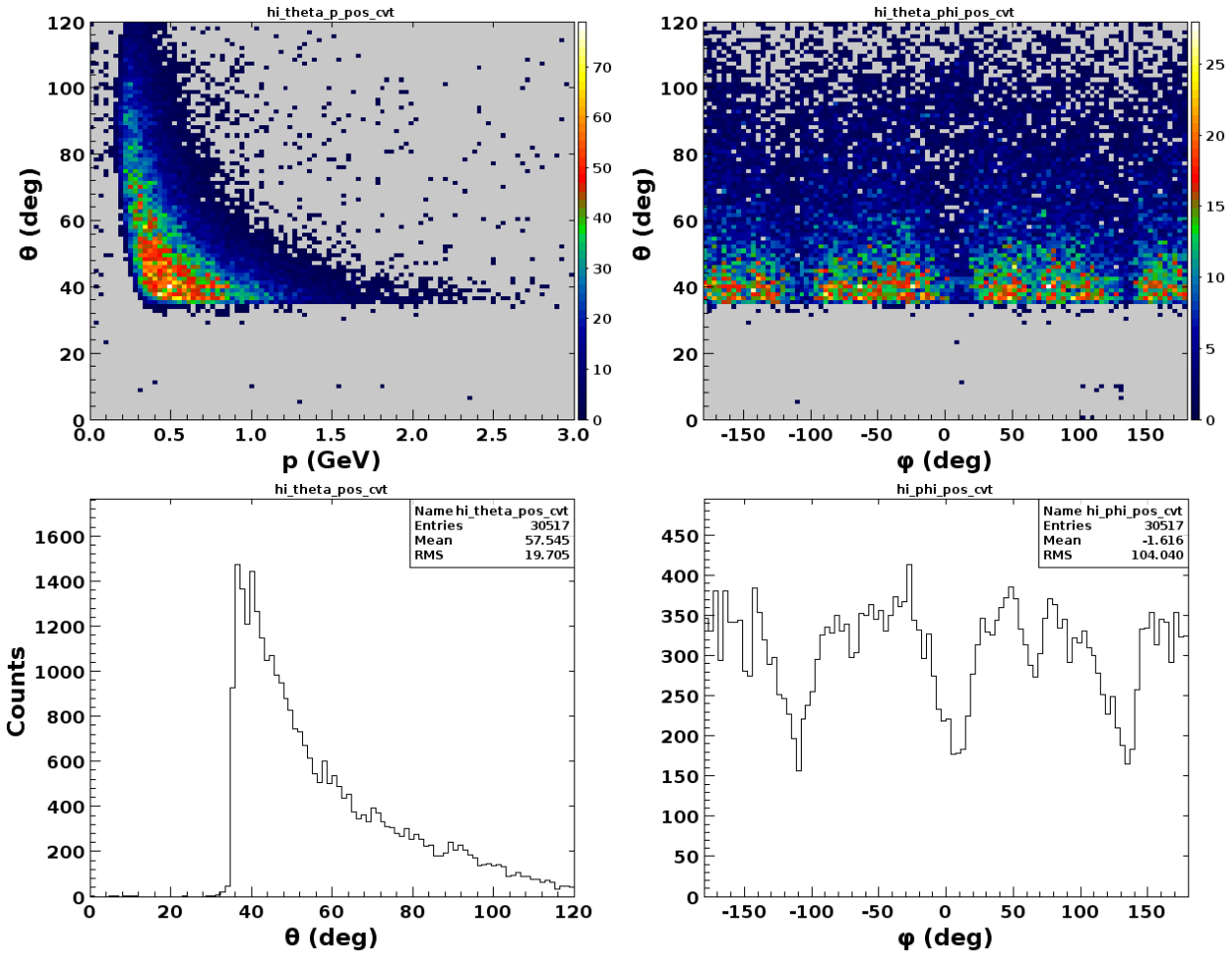


Figure 26. CTOF Vertex Positions for Positive Particles.

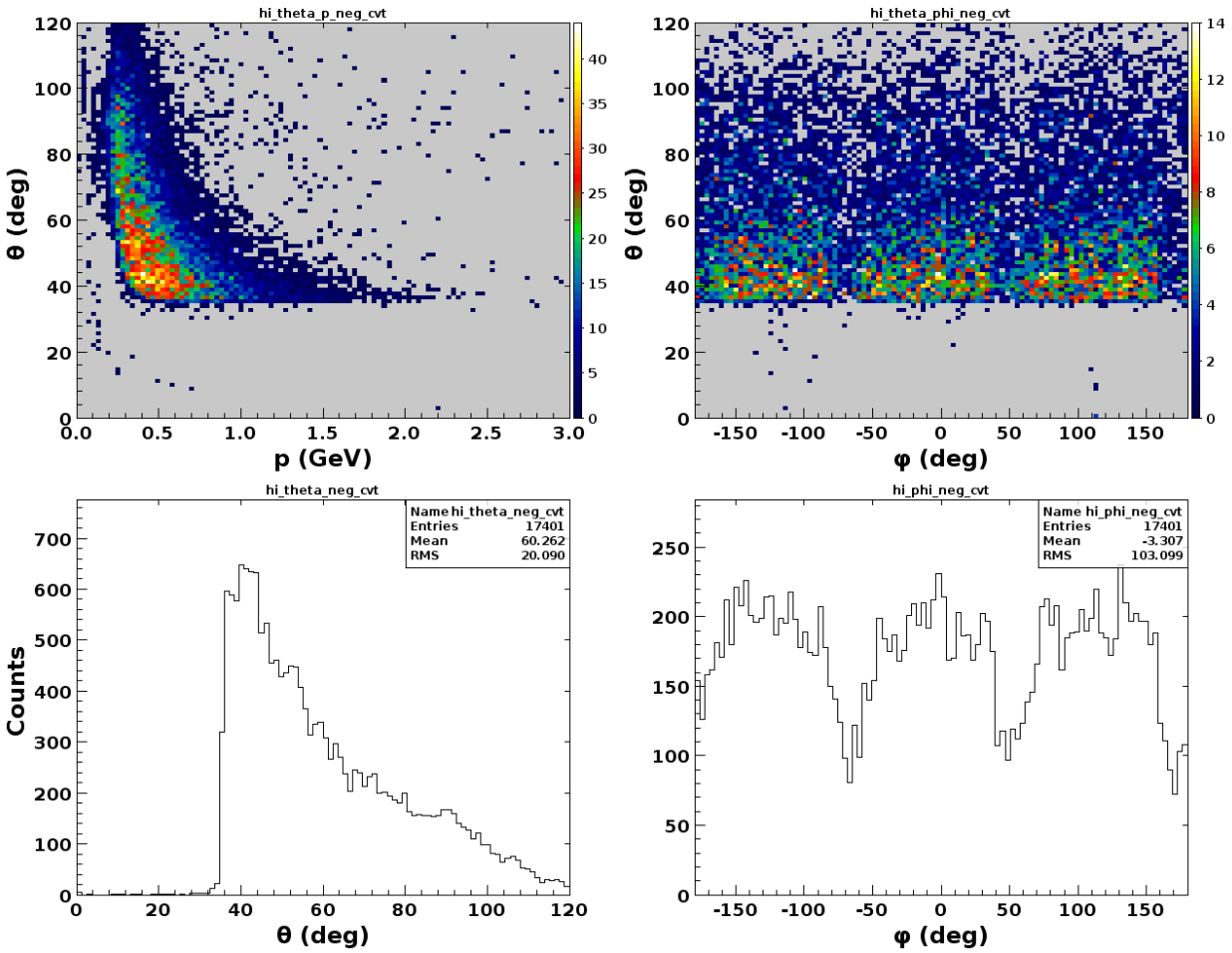


Figure 27. CTOF Vertex Positions for Negative Particles.

4.4 Forward Detector and Central Detector Vertex Comparison

CVT and forward detectors subtend differing layers of ϕ and must agree on the vertex location for particle reconstruction. Scattered particles from the same events within the target often interact with both detectors separately, and so by tracing these events back to the same vertex, one can calculate the differences in vertex alignment between each detector.

The CVT and forward vertex locations are not quite in agreement. Events with two tracks, one leading into the CVT and the second leading to the forward detector sectors can be used to correlate the reconstructed vertex locations.

The CVT vertex distribution is wider than the forward detector, probably due to the geometry of the CVT counters not being correctly calibrated. Differences in the actual CVT geometry, along with the assumptions that are made with the hit point locations within the scintillation bars. The

downstream end of the bars are curved leading to path length errors up to 10% of the actual pathlength.

These plots show the correlation between the reconstructed vertex locations of the CVT and forward detector for a vertex location ideally centered at $v_z = 0\ cm$.

Banks in the CVT and forward detector both must contain, at least one particle of the same charge associated the same event. If both detectors were aligned, each particle track in the same event can be traced back to the vertex at $v_z = 0\ cm$. Firstly, particles with negative charge detected in the CVT were compared to particles with negative charge detected by the forward detector contained within the same event, first by selecting the negative particles in the CVT, and then by looping over all negative particles detected by the forward detector. Each negative particle associated with the same event in the CVT would be selected and compared to each particle in the forward detector by looping over the rows in the CVT bank and selecting particles with charge less than zero. The same procedure was done for comparing positive tracks, except the particles with charge greater than zero were selected and compared.

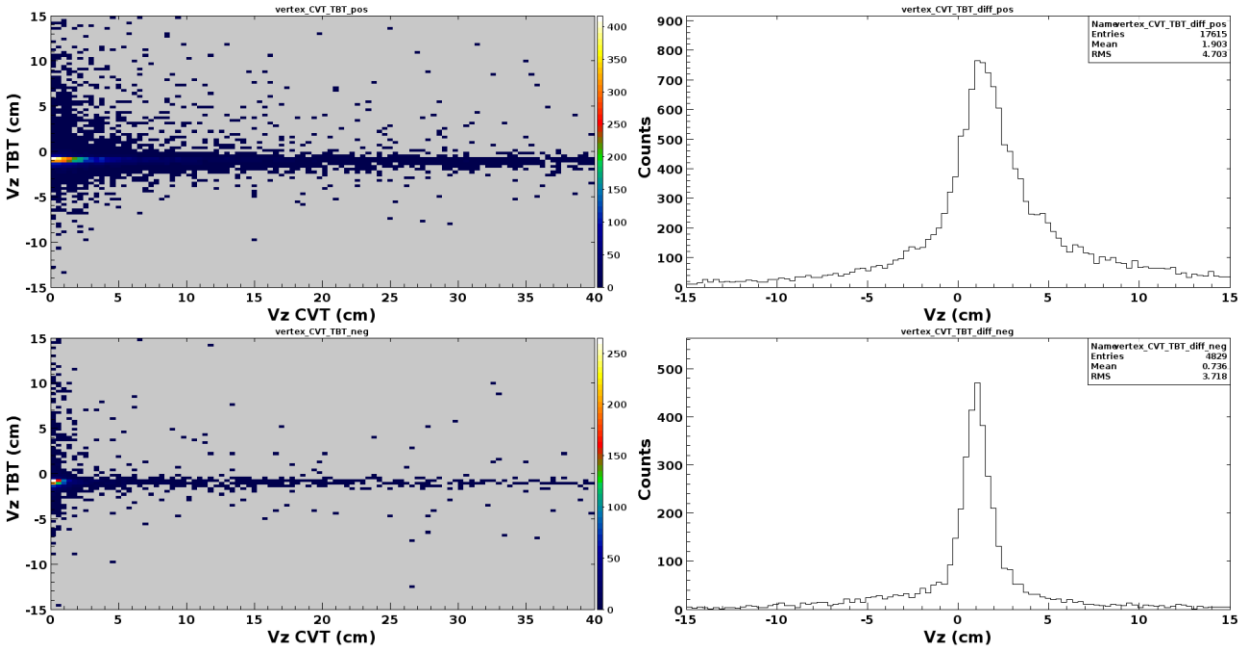


Figure 28. CTOF Vertex Comparison.

Offsets of roughly 1.9 cm for positive tracks and 0.7 cm for negative tracks were found as the mean of the distribution without cuts in momentum or angle. Both plots show similar wide

distributions along the x axis indicating a poorer resolution in CVT as compared to the forward detector.

Below is a table comparing the mean vertex locations for the CVT and forward detector detectors and the Vertex difference for each charge. No cuts were used in this data, so the values in columns one and two represent the mean for all particles.

Table 4. Vertex Comparison.

Charge	CVT Vertex (cm)	Forward Detector Vertex (cm)	Difference (cm)	Vertex Comp.	Percent Difference
Pos	-0.819	1.181	2.000	1.903	4.97%
Neg	-0.722	0.332	1.054	0.736	35.53%

Percent differences in the right most column show the disagreement between the two methods used for determining vertex locations.

4.5 Forward and Central Vertex Comparison with Extended Target

With the extended target, the plots show scattering pattern as expected with the extended target but with similar long smearing pattern as seen with respect to the CVT. This is because the forward detectors resolution is lower than that of the forward detector.

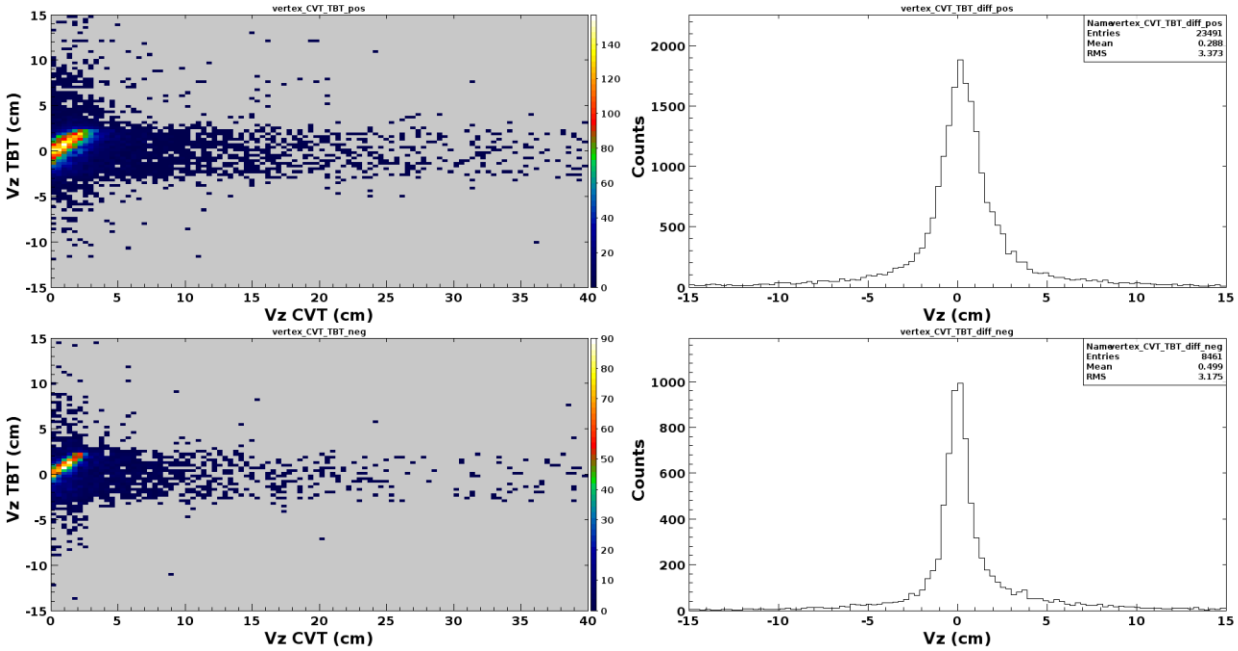


Figure 29. Vertex Comparison for Extended Target.

Histogram shows mean differences of 0.288 cm and 0.499 cm for positive and negative tracks respectively.

Table 5. Vertex Comparison for Extended target.

Particle Track	CVT Vertex	Forward Detector Vertex	Difference	Vertex Comp	Percent Difference
Pos	-0.142	0.189	0.331	0.288	13.89337641
Neg	-0.174	0.286	0.46	0.499	8.133472367

4.6 MC Comparison

Monte Carlo simulations can reveal the ideal detector resolutions and defects in the geant4 geometry, as well as describe physical interactions within the detector not accessible during experiments. The geant4 Monte Carlo algorithm using simulated targets and detector geometry to achieve this. Accuracy depend on whether the simulated reconstructed event matches with what is expected, which can be achieved during these simulations since every parameter of the particle jets are known and can be compared. Obviously, the reconstructed event with respect to

the simulated detector geometry cannot be reconstructed perfectly, just as in the real world, so this provides a way probe the resolution of the actual detector and its reconstruction algorithm. In the figures 31, and 32 we see this comparison between the reconstructed particle parameters and the expected values from the actual simulated particle kinematics.

Later versions of the reconstruction algorithm improve the path length fits along the SVT, and calorimeters, as well as improvements in matching the actual uneven geometry of the detector, such as matching the misalignment of the forward detector, since the goal of the simulation is to match the real world as much as possible. The kinematics in the following graphs show these differences among the simulated and reconstructed particles using the geant4 geometry.

In short, the standard deviations must be as narrow as possible in order to achieve the beast accuracy. A typical generated particle event in the bank takes the following form:

Table 6. Generated Event Bank.

pid :	11	2212	211	-321	321	-211
px :	0.1495	0.0163	0.1132	-0.1758	-0.0317	-0.0717
py :	0.6836	-0.1178	0.1265	-0.2809	-0.4308	0.0174
pz :	4.3412	1.6636	1.3915	2.6486	0.1586	0.4452
vx :	-0.0016	-0.0016	-0.0016	-0.0016	-0.0016	-0.0016
vy :	0.0028	0.0028	0.0028	0.0028	0.0028	0.0028
vz :	0.1665	0.1665	0.1665	0.1665	0.1665	0.1665
vt :	124.25	124.25	124.25	124.25	124.25	124.25

The PIDs in the first row correspond to Particle Identification numbers set forth by the Monte Carlo Particle Numbering Scheme. Here we see kaons (321, and -321), electrons (11) and photons (2212). Each particle has a vertex position, momentum, and event start time associated with the it. Each simulated event has an electron in the first column associated with it, as this electron is the spectator electron associated with the beam and is used to find coincidences. These can be converted into spherical coordinates and compared to the reconstructed values measured by the banks CVTRec::Tracks and TimeBasedTrkg::TBTracks, which is the reconstructed CTOF and forward detector banks respectively.

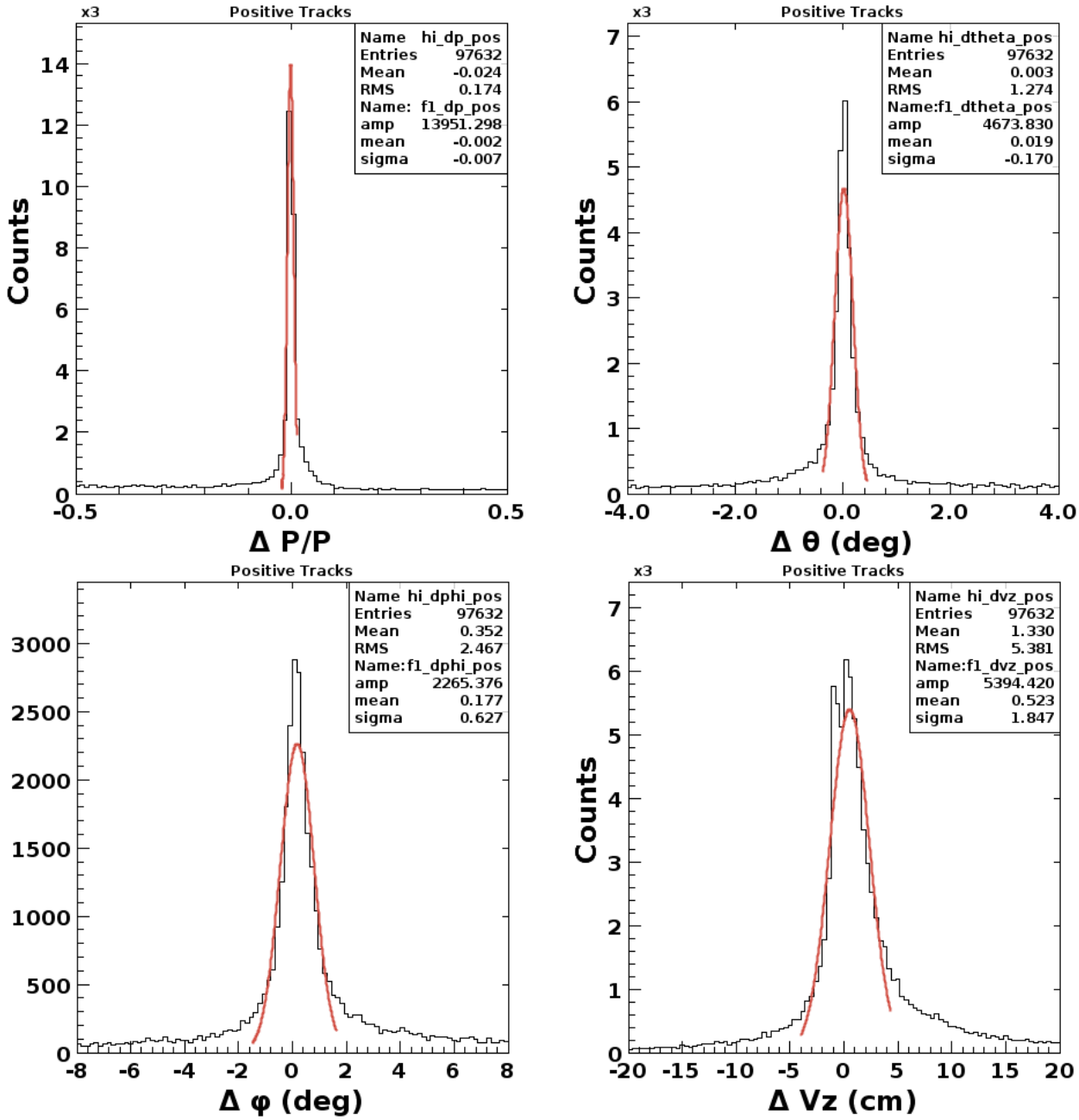


Figure 30. MC Comparisons for Positive Particles.

Figure 30 shows the difference between the reconstructed momentum and generated momentum for positive particles. Resolution here shows .7% difference, although ideally this should be below .2%. This is calculated using a gaussian fit without cuts. Figure 30 (upper right) shows differences in the angular distribution θ, which was calculated to be delta θ = .17 degrees. Similarly, with angle ϕ, the difference was calculated to be ϕ = 0.627 degrees. The vertex location, tends to be off by 0.523 cm, with a standard deviation of 1.847 cm.

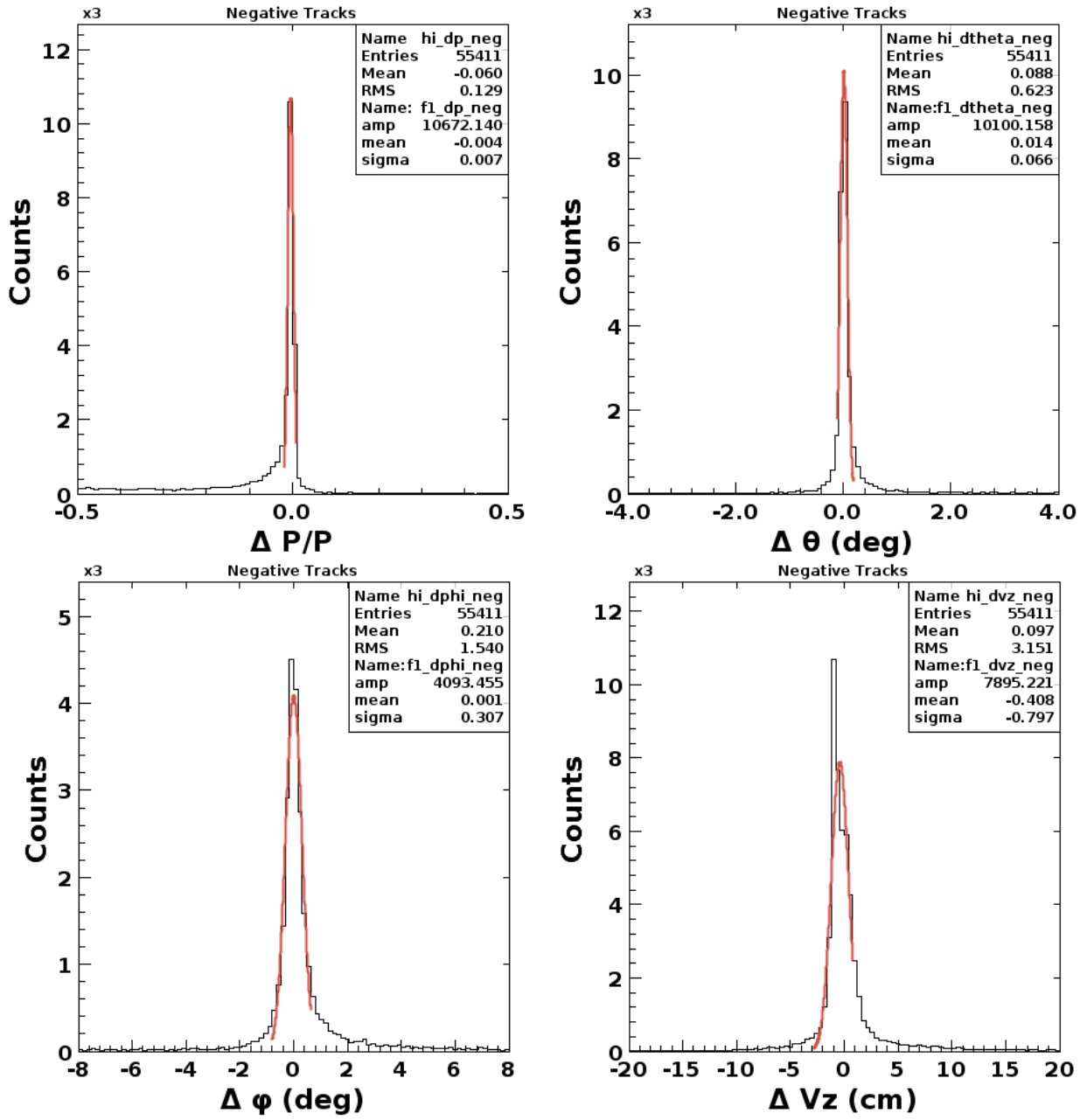


Figure 31. MC Comparisons for Negative Particles.

Figure 31 shows the difference between the reconstructed momentum and generated momentum for negative particles. Resolution here shows .7% difference, although ideally this should be below .2%. This is calculated using a gaussian fit without cuts. Figure 31 (top right) shows differences in the angular distribution θ, which was calculated to be delta θ = .066 degrees, smaller than the positive tracks by under a half. Similarly, with angle ϕ, the difference was

calculated to be $\phi = 0.307$ degrees, once again half the σ of the positive particles. The vertex location, tends to be off by -0.408 cm, with a standard deviation of 0.797 cm.

4.7 FTOF Detector

Positive tracks desposited across the FTOF scintillations arrays are plotted in this section. At this point along the entire clas12 detector, all particles which reach the FTOF arrays are either kaons, pions, muons, photons, or electrons.

Figure 32 below shows the distribution of positively charged particles deposited on the forward detector with respect the total momentum. A typical landau distribution results and should contain mostly protons, K^+, and π^+. The mean momentum of the distribution of positive particles was 1.684 GeV/c among approximately 78,000 entries. Particles in this case showed scattering angles with a mean distribution of $\theta = 17.624$. Particles showed no preference towards scattering in the azimuthal direction as shown in figures 36. The scattering pattern seen in these plots is due to the characteristics of the magnetic field. Each hot spot in figure 33 is the central section of the FTOF panel arrays. Detector coverage is cut off at $\theta = 5$ so this region should be empty.

The reconstructed vertex positions measured here showed a mean around $v_z = 0.200\ cm$, with a standard deviation $\sigma = 0.853\ cm$. This is an outstanding result, since this results in a timing resolution of about 30 ps.

The distribution of particles across the θ direction is shown in figure 32. This shows a slight preference for high energy particles towards the low scattering angles, but overall the deflection of positive particles don't seem to be as affected by its momentum.

χ^2 is used here to test frequency of deviations between the expected track and observed track locations. χ^2 distribution in this case can be found in figure 33. Here, the mean χ^2 value was 26.856. χ^2 values as shown with respect to the vertex location are shown in figure 33.

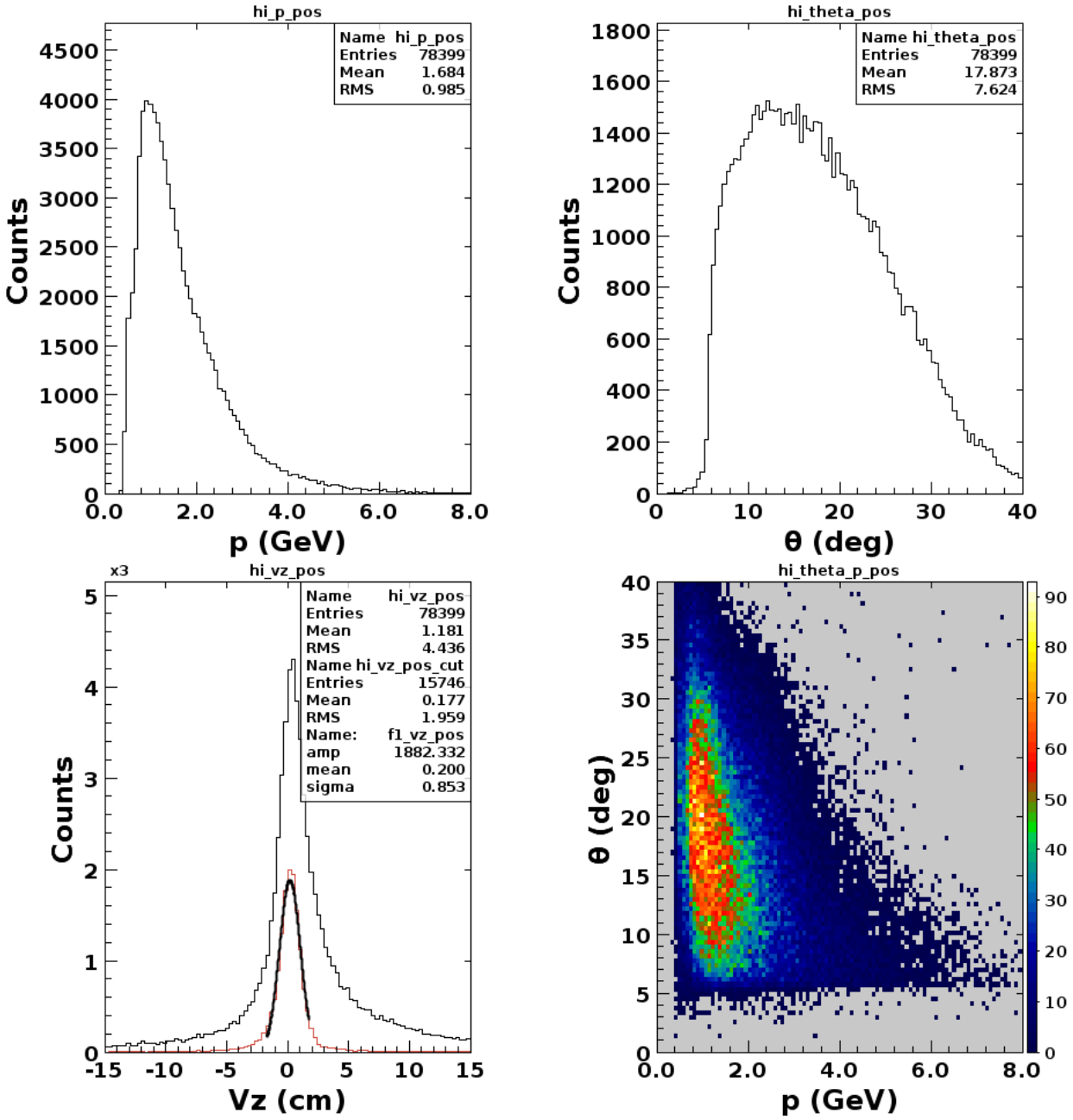


Figure 32. FTOF Detector Plots for Positive Particles.

Figure 31 below shows the distribution of negatively charged particles deposited on the forward detector with respect the total momentum. In contrast to the typical landau distribution as was the case with the positive particles, the distribution here shows more particles clustered towards higher momenta. These particles are the beam electrons scattered at low angles through the detector. The rest of the distribution should contain mostly spectator electrons, K^-, and π^-. The mode, or peak seen in this histogram, is around 1 GeV/c. The second histogram from the top left

shows the angular distribution of the scattered negative particles, which shows a prominent peak around $\theta = 10$ of mostly beam electrons. Once again, the scattering pattern shows no preference in the azimuthal direction.

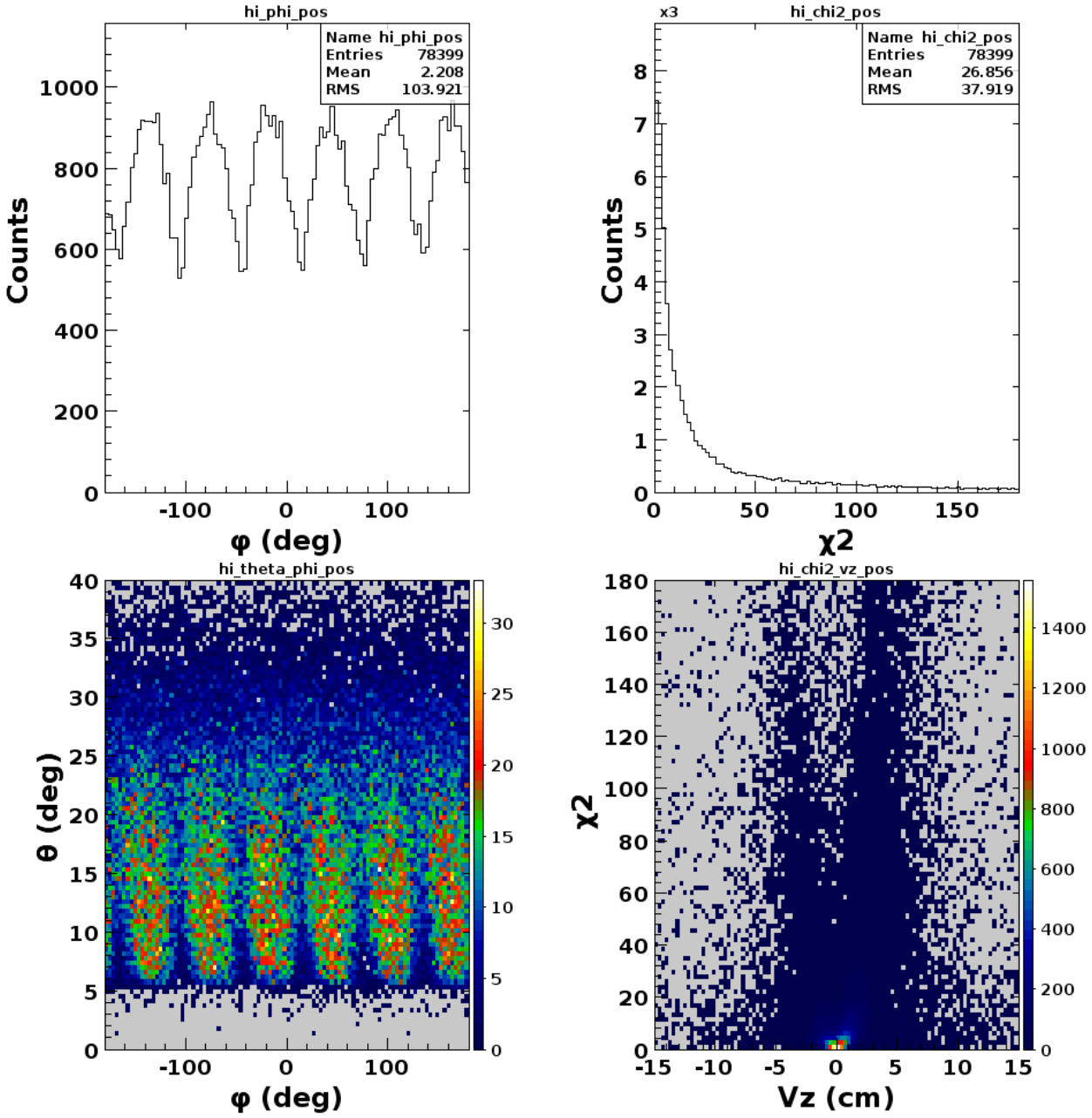


Figure 33. FTOF Detector Plots for Positive Particles.

The reconstructed vertex positions measured here showed a mean around $v_z = 0.210\ cm$, with a standard deviation $\sigma = 0.610\ cm$. Once again this is an excellent result.

The distribution of particles across the θ direction with respect to θ is shown in figure 34 (upper right). This shows a strong preference for high energy particles towards the low scattering angles. At higher scattering angle, we see a higher concentration of scattered particles.

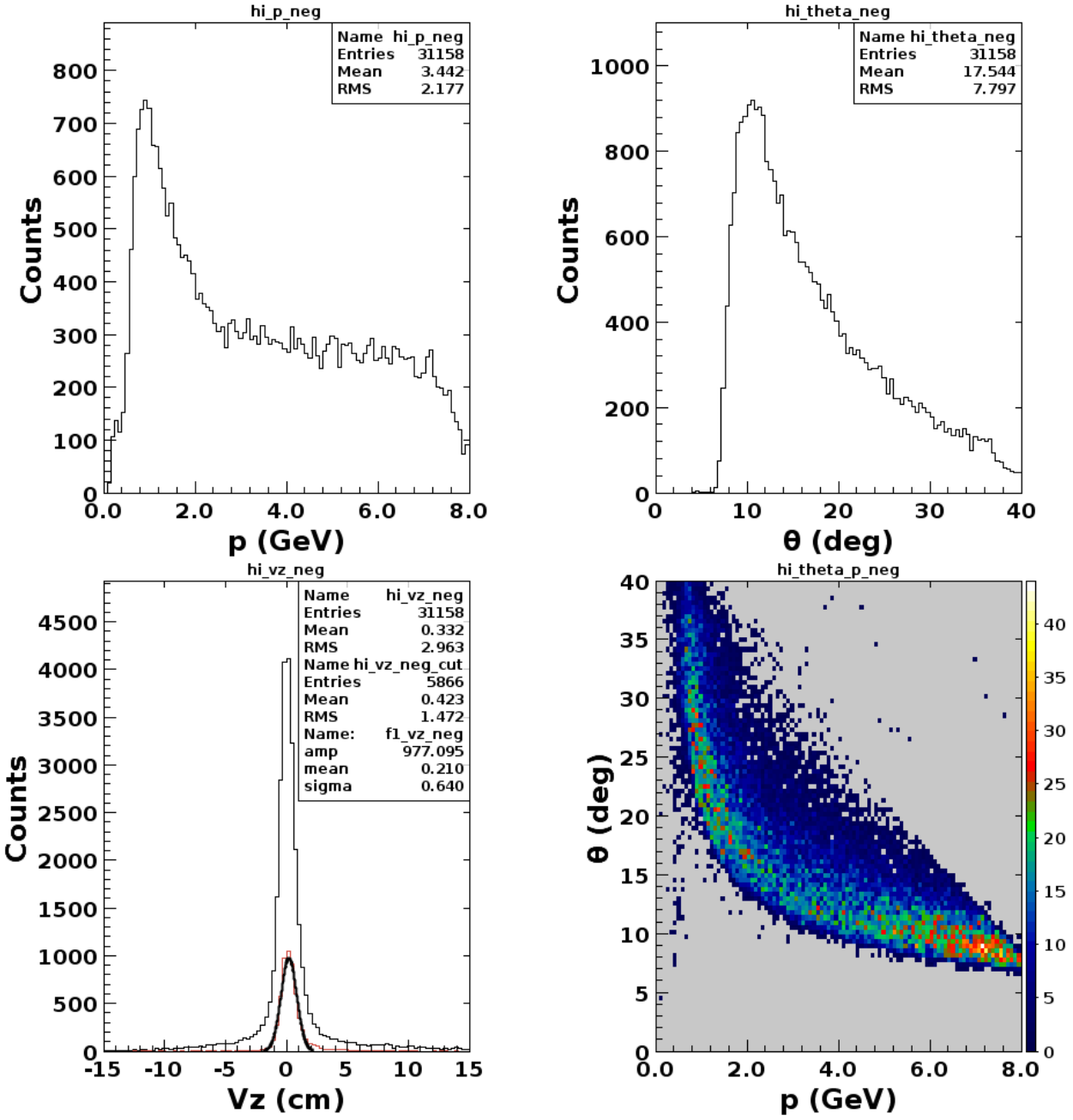


Figure 34. FTOF Detector Plots for Negative Particles.

4.8 For the Extended Target

4.8.1 FTOF For the Extended Target

Figure 35 below shows the same distributions of positively charged particles with the extended target for the FTOF. Distributions with respect to the momenta as above should not change, as is the case. Vertex distribution shows the extended vertex, with a mean of $v_z = 0.189\ cm$ for all positive particles.

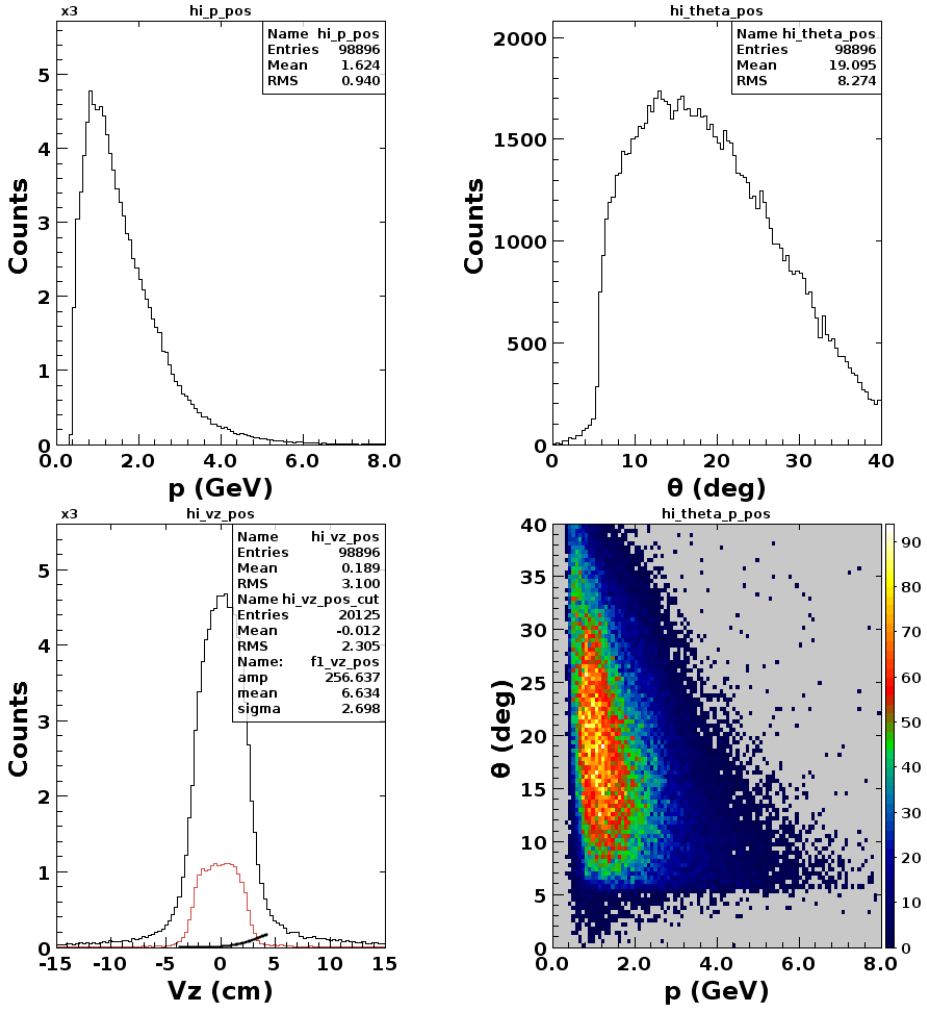


Figure 35. FTOF Detector Plots for Positive Particles with Extended Target.

The distribution of particles across the θ direction are shown in figure 35 (upper right). This shows the same slight preferences for high energy particles towards the low scattering angles, but with a small flare below θ =5 degrees.

The mean χ^2 value was 9.621. This is an improvement from the un-extended target, indicating that the inclusion of the extended target greatly improves track reconstruction.

Figure 36 below shows the same distributions of negatively charged particles with the extended target for the FTOF. Distributions with respect to the momenta show an additional rise in particles with higher momenta around p = 6 GeV/c. Vertex distribution shows the extended vertex, with a mean of $v_z = 0.286\ cm$ for all negative particles. Angular distributions with respect to momentum shown below, show an even greater clustering of particles with lower energies, due to more entries than the previous plots.

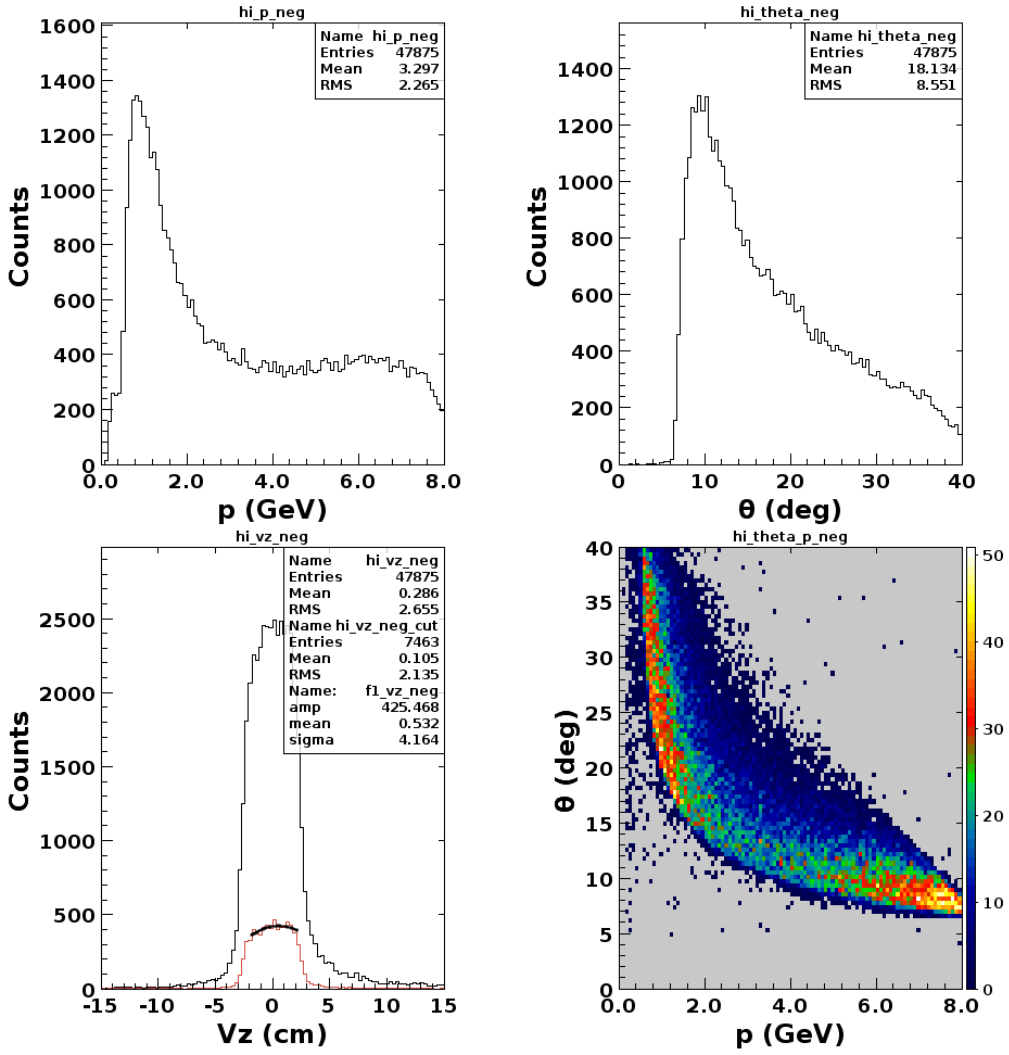


Figure 36. FTOF Detector Plots for Negative Particles with Extended Target.

In this case the mean χ^2 value was 8.644. This is an improvement from the un-extended target, indicating that the inclusion of the extended target greatly improves track reconstruction.

4.8.2 CVT and Forward Vertex Distributions for an Extended Target

The CVT vertex distributions for an extended target were plotted for positive and negative particles. These plots are the same as above but with the extended target of 5 cm. Here we can see the effect of the extended target clearly. Notice how the vertex tends to skew towards the downstream end of the target:

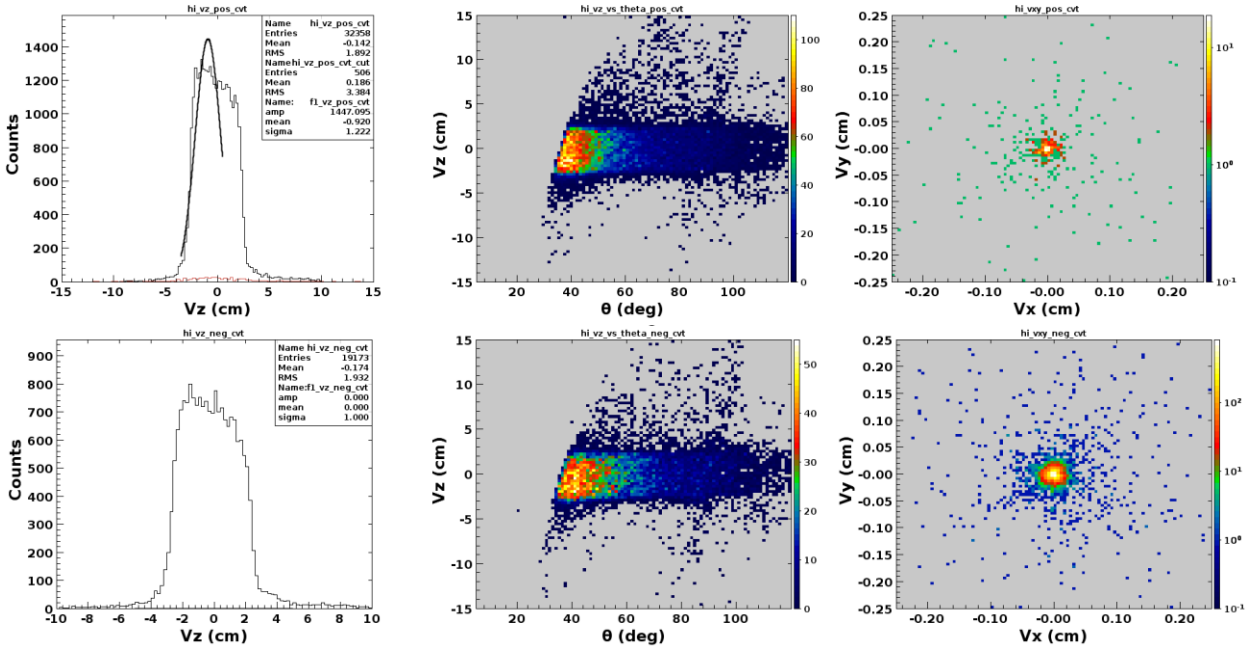


Figure 37. CVT Detector Plots for Negative Particles with Extended Target.

4.8.3 Forward and Central Vertex Comparison with Extended Target

With the extended target, the plots show scattering pattern as expected with the extended target but with similar long smearing pattern as seen with respect to the CVT. This is because the CVT's resolution is lower than that of the forward detector.

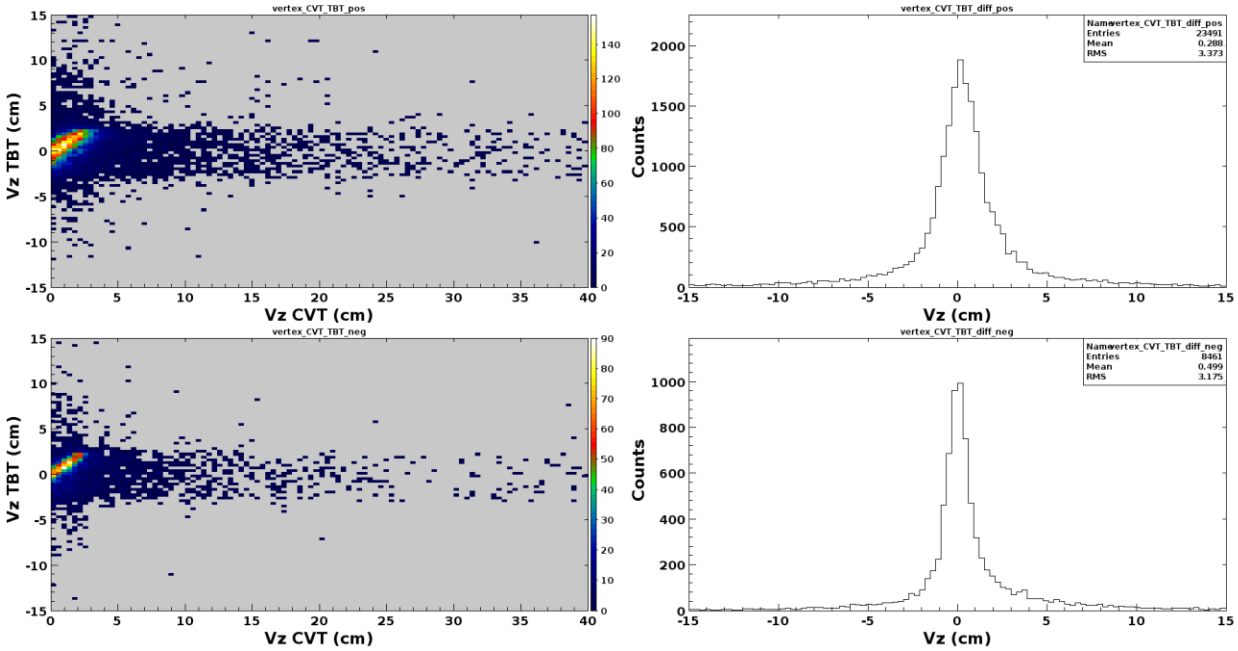


Figure 38. Vertex Comparison for Extended Target.

Histogram shows mean differences of 0.288 cm and 0.499 cm for positive and negative tracks respectively.

Table 5. Vertex Comparison for Extended target.

Particle Track	CVT Vertex	Forward Detector Vertex	Difference	Vertex Comp	Percent Difference
Pos	-0.142	0.189	0.331	0.288	13.89337641
Neg	-0.174	0.286	0.46	0.499	8.133472367

5. Charged Particle Identification using the TOF system

Charged particles can be identified using the particle's time of flight and momentum measurements taken from CLAS12's TOF system. Experimental uncertainties and the accelerator's RF signal must be accounted for so the identification can be optimized. [4]. Fluctuations of propagation time of the trigger signal leads to large event start time uncertainties.

The event start time contains large uncertainties due to the fluctuations of propagation time of the trigger signal, and so event analysis should be used in leiu of time information for determining of event start time used for finding t_i^{TOF}.

In electron scattering, every event is required to have an electron. The event canidate is chosen by drift chamber hit-based momentum reconstruction, shower profile and energy deposition in the forward electromagnetic calorimeter, and hits in the Cherenkov and scintillator counters. The event start time t_{St}, is the difference between the the measured electron hit time t_e^{SC}, and ToF of the electron:

$$t_{St} = t_e^{SC} - \frac{R_e}{v_e}$$

Where R_e is the electron path length to the sciltilator plane, and v_e is the electron velocity.

Events in which the first particle in the REC::Particle bank was an electron detected in the forward Detector were chosen. Next, the FTOF time and path of the electron was found by looping over the REC::Scintillator bank and selecting the entry with the electron. Hits from FTOF panel 1B were used, since the particle travels through both panels and only one is needed.

$$t_{RF} = n \times \delta t_{RF}^c + t_{RF}^o + t_{St}$$

Where $\delta t_{RF}^c = 2.004\ ns$ is the RF structure constant related to the time interval between bunches in the electron beam, and t_{RF}^o is the parameter related to the electron ToF time between the RF separartor and the target. Note that the bunch length is only a few picoseconds, so this parameter can be neglected in this derivation. The parameter n is an integer with a random value, but whose value is limited by the trigger time window. The beam RF time structure as seen with from the difference between start time and RF time (figure 39). The peaks here are 2.004 ns apart, and this cooresponds to 3x the bunch width from the acclerator.

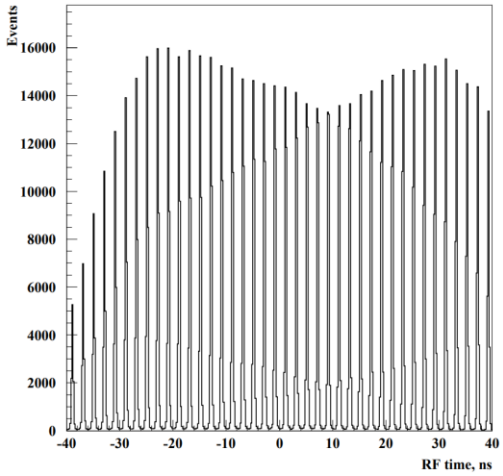


Figure 39. RF Start Time with Respect to Event Start Time. Image Credit [5].

Each peak is folded together by taking the the modulus of $t_{RF} - t_{St}$ with δt^c_{RF}. Centering the resulting distribution results in the corrected time:

$$\Delta t_{Off} \;=\; MOD(t_{RF} - t_{St}, \delta t^c_{RF}\,) - \frac{\delta t^c_{RF}}{2}$$

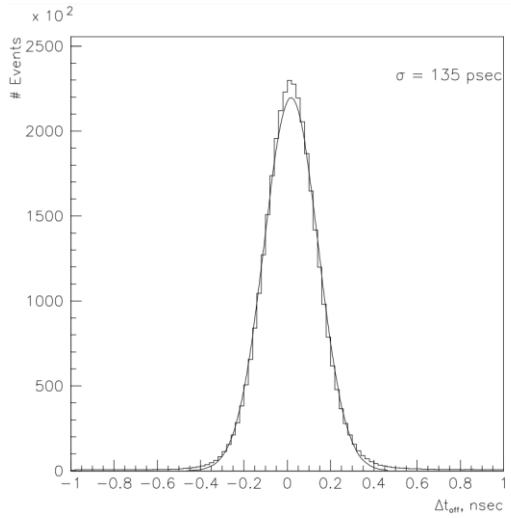


Figure 40. The Time Δt_{Off}. Image credit [5].

The parameter t^o_{RF} will be varried so the distribution Δt_{Off} is centered at 0. This correction is applied to the start time t_{St}.

The velocity of the particle β, is related to the electron path length and ToF by:

$$\beta_i \;=\; \frac{R_i}{c \times t_i^{ToF}}$$

Here, the particle's time of flight is the difference between t_i^{SC}, the scintilator hit time and the RF corrected srart time $t_{st} + \Delta t_{Off}$.

See figure 41 for identification of protons kaons and pions. Here we can distinguish the common positively harged particles present in these scattering experiments, with heavier particles forming the lower bands, and the lighter pions forming the upper band. The velocities of these pions are close to the speed of light ($\beta \approx 1$). The distributions for the negatively charged particles will not be distinguished, since the number of events displyed here is too small to identify anything other than electrons.

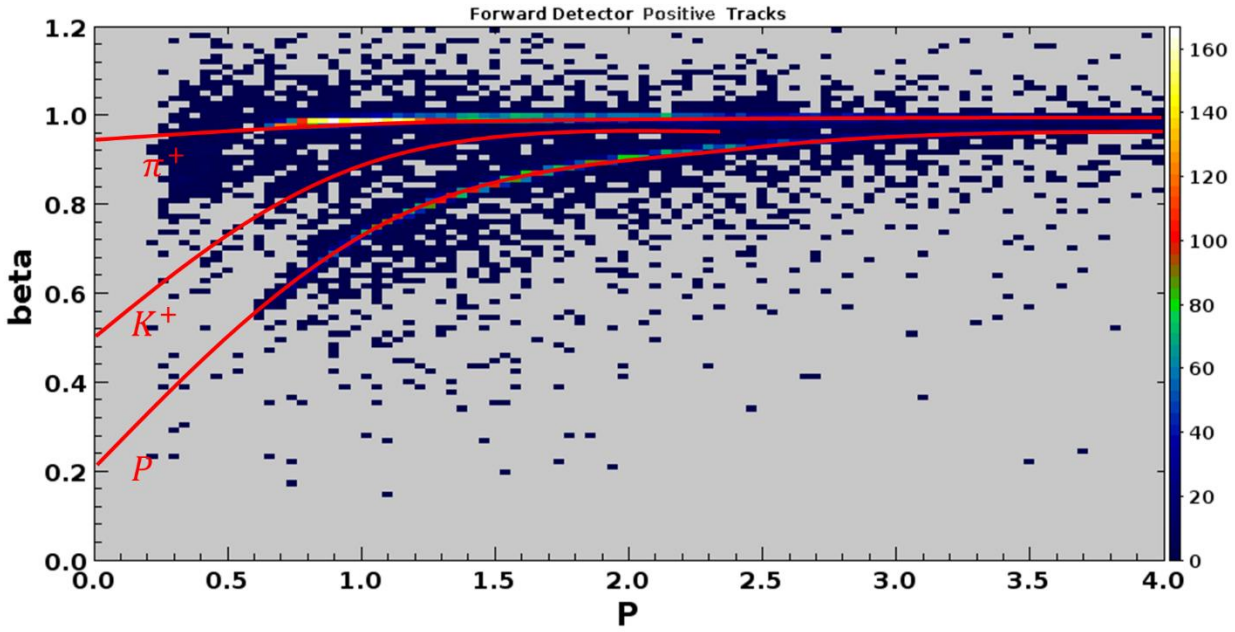


Figure 41. Positive Particle Identification using Velocity vs. Momentum.

5.1 Without Vertex Correction

Using the β from reconstruction, the β vs. momentum plots were made for the forward TOF detector (see figures 45). The histograms are shown next to the scatterplots as well. β from reconstruction shows a strong band of pions, or muons with velocities very close to the speed of light, at low momentum. Casual explanations for this indicate errors in the reconstruction for particles at low momenta.

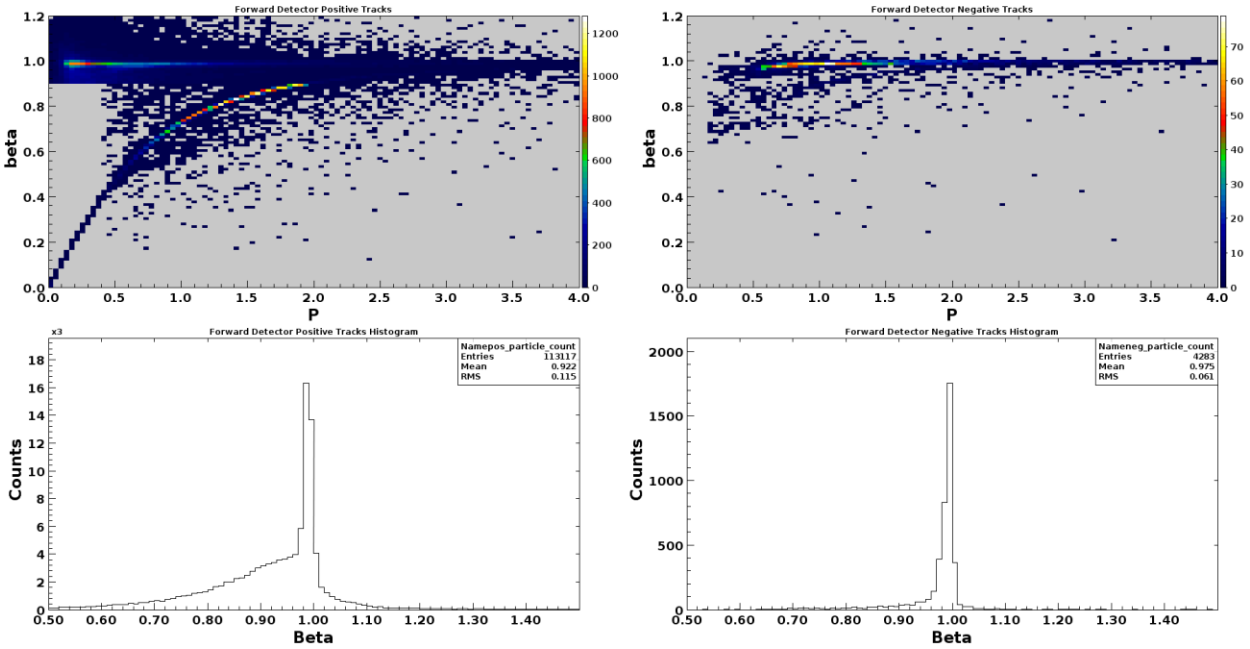


Figure 42. Negative and Positive Particles using Reconstructed β.

Below we see the beta calculated from scratch. The positive particles distributions do show bands of protons, kaons, and pions, in order from most massive to least massive. The histogram here shows proportionally less massive particles to lighter particles than the reconstructed beta plots indicate (see figures 46 and 47).

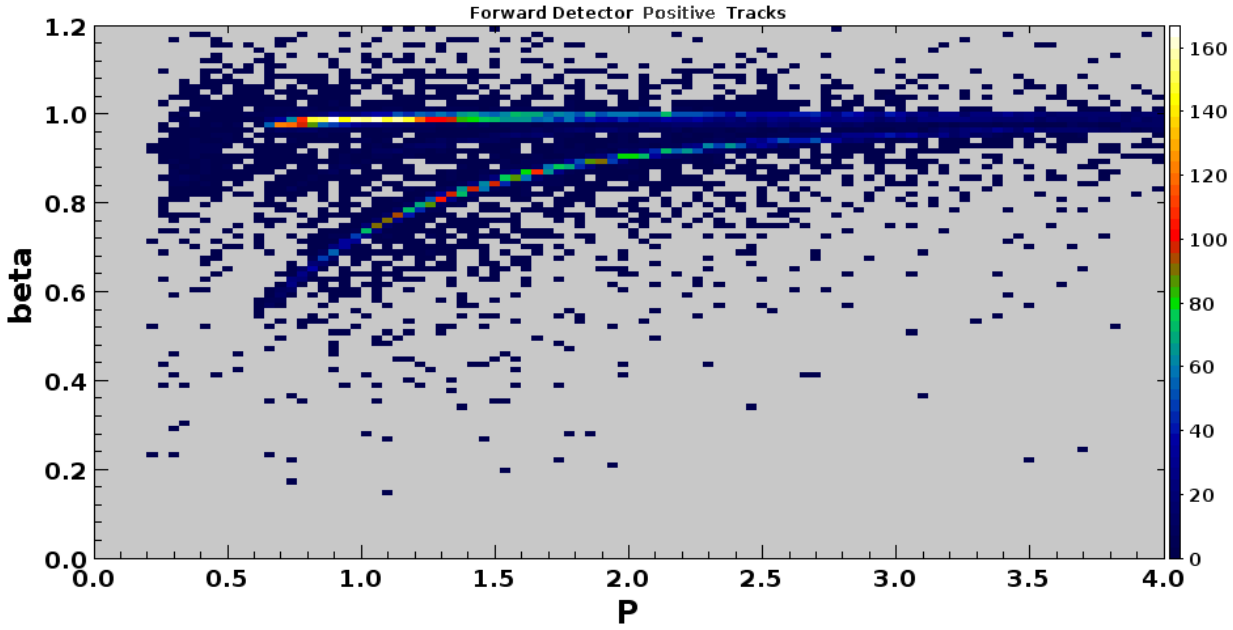


Figure 43. Positive Particles using Calculated β.

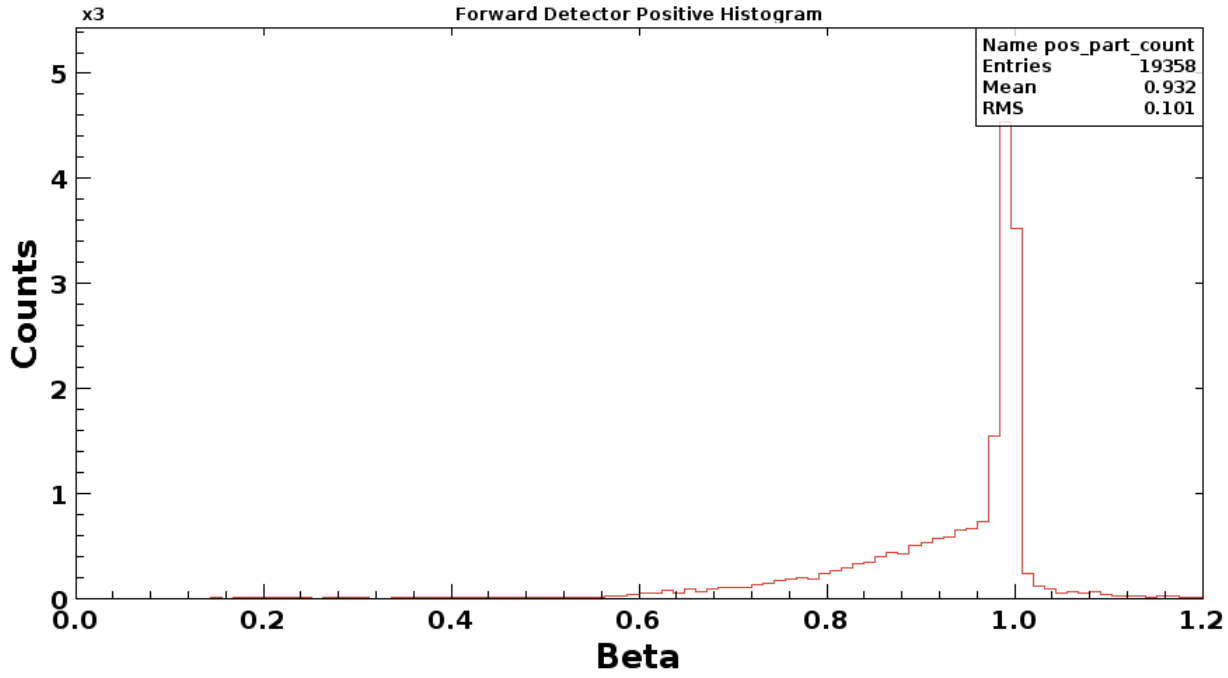


Figure 44. Histogram, Positive Particles using Calculated β.

Negative particles detected in the forward detector mainly show the band from forward scattered electrons, with a possible band of negative pions. Statistics will have to be improved for more particles to be resolved.

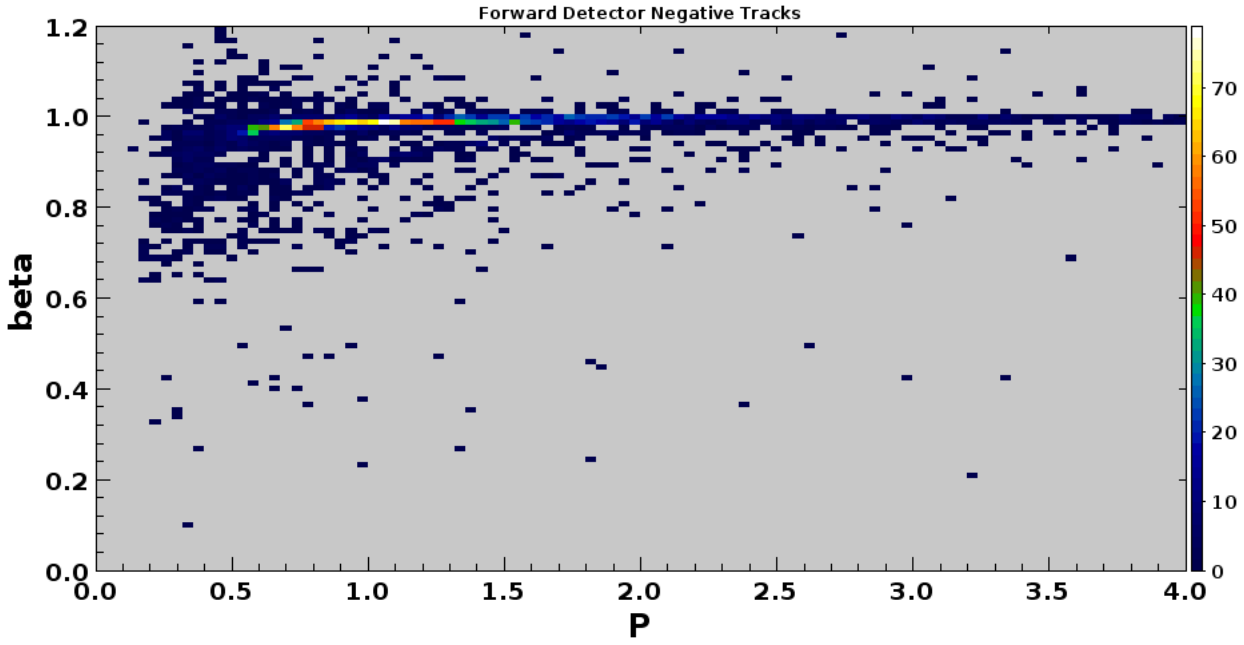


Figure 45. Negative Particles using Calculated β.

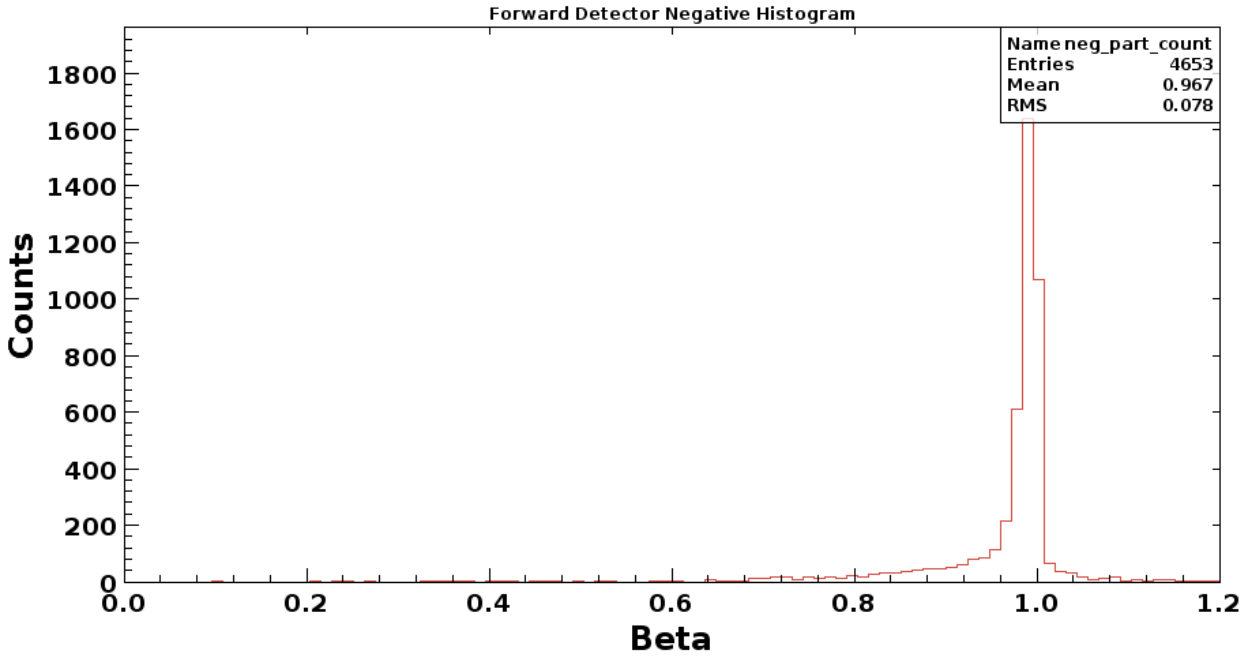


Figure 46. Histogram, Negative Particles using Calculated β.

5.2 With Vertex Correction

With the vertex correction, the scatter plot (figure 47) and histogram (figure 48) reveal a tighter distribution around the proton and pion bands. This shows that the vertex correction does improve the resolution of the detector.

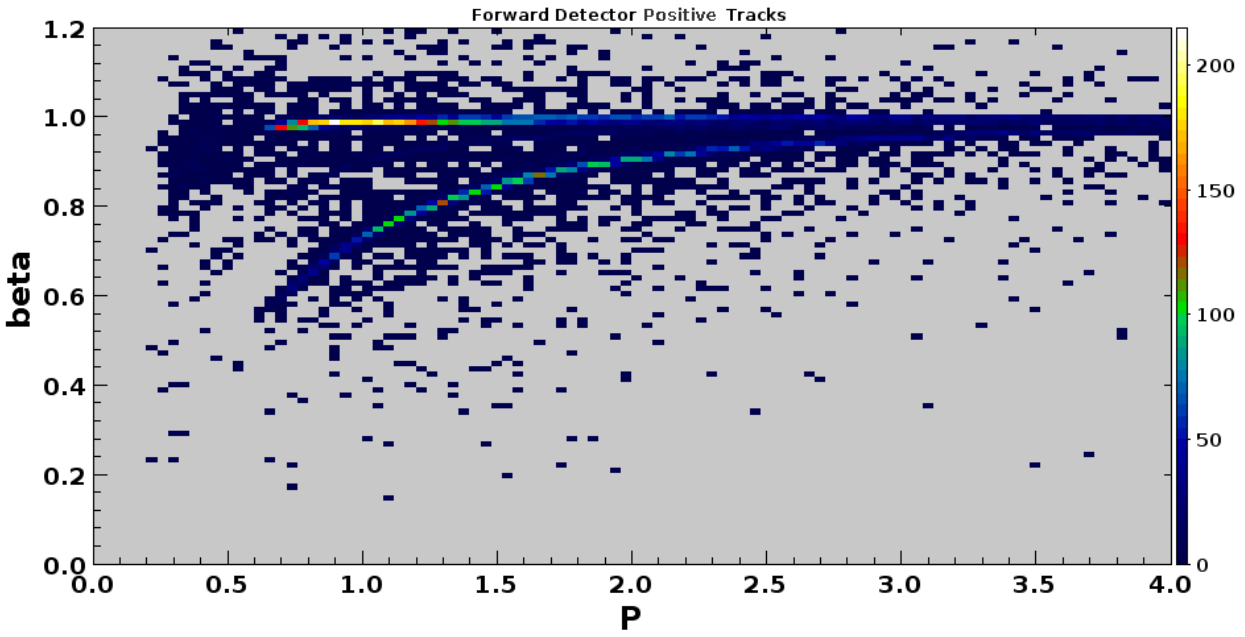


Figure 47. Positive Particles using Calculated β, with Vertex Correction.

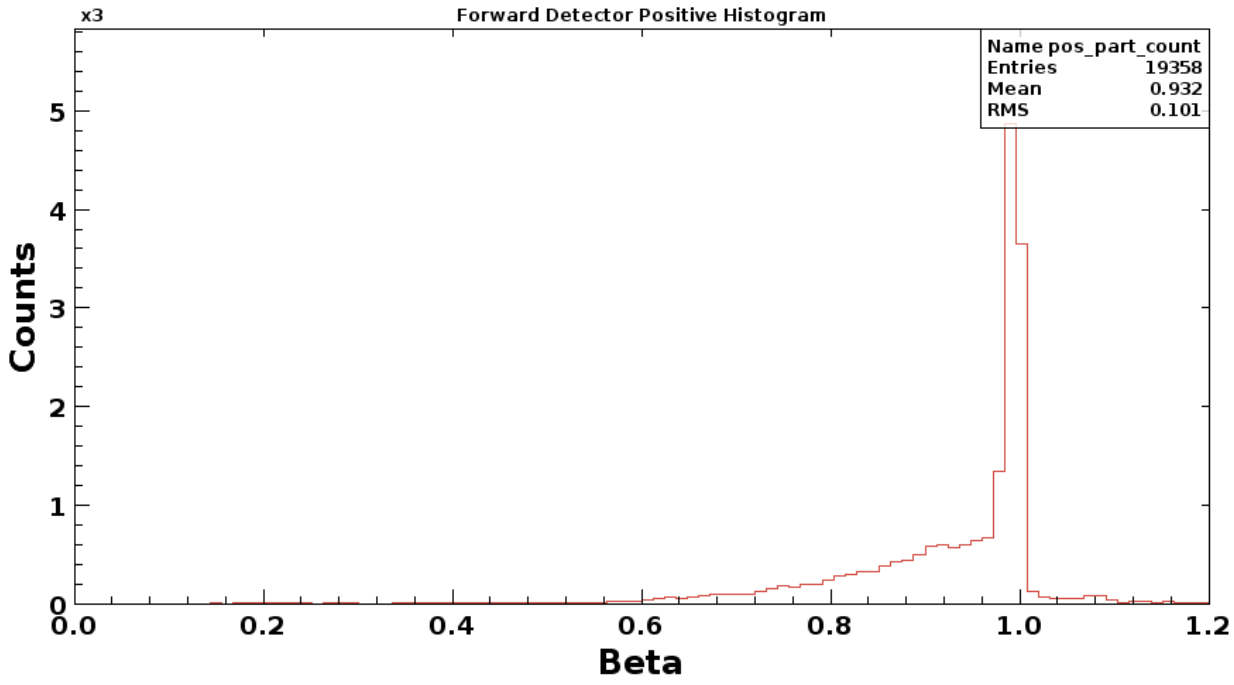


Figure 48. Histogram, Positive Particles using Calculated β, with Vertex Correction.

Similarly, to the positively charged distribution, the negatively charged particle plots show improved resolution as well. A slightly more noticeable kaon band might be seen now, but once again, more events are needed in order to resolve more massive particles.

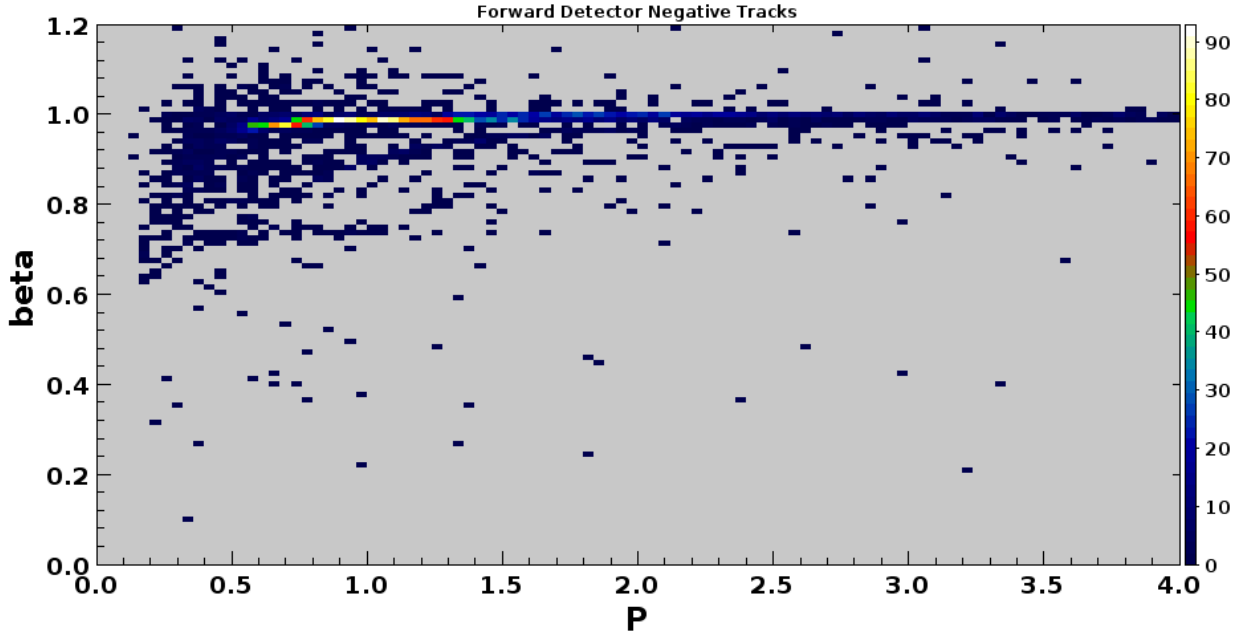


Figure 49. Negative Particles using Calculated β, with Vertex Correction.

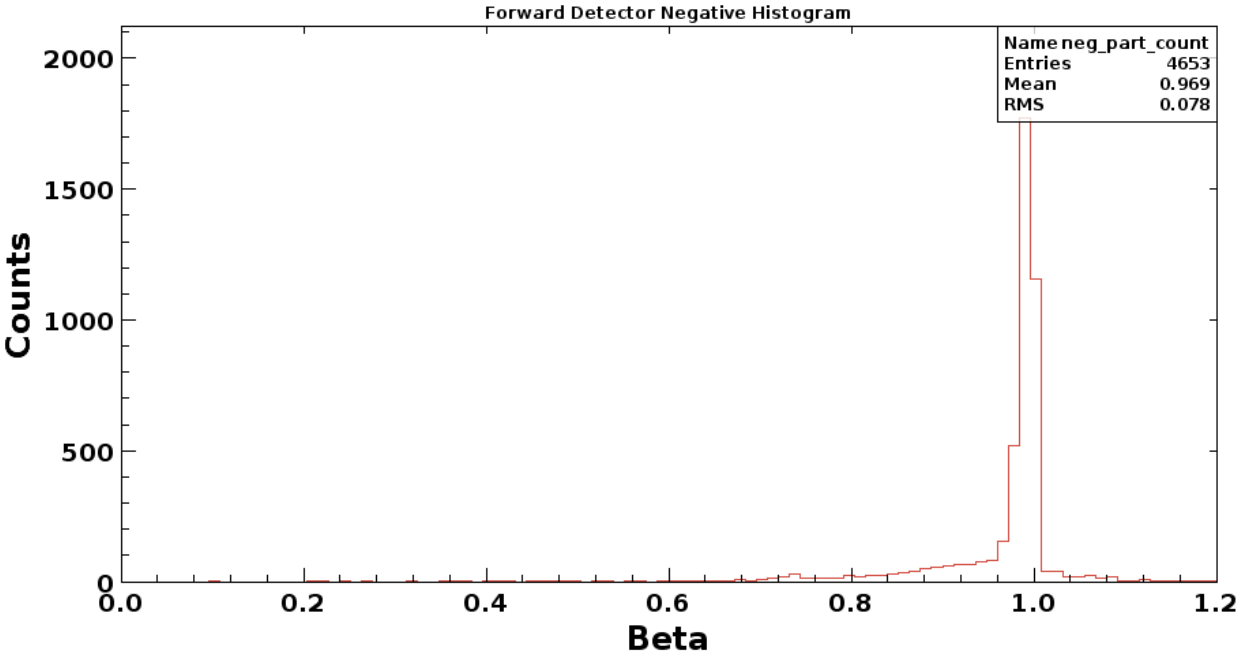


Figure 50. Histogram, Negative Particles using Calculated β, with Vertex Correction.

5.3 A Note about Particle Resolution.

A study will need to be completed involving the resolution of the TOF system with respect to the particle identification mentioned earlier. The standard deviations of each of the bands in the plots in this section will have to be measured using a gaussian fit around the energies mentioned in the specification. This will not be included in this paper.

6. Conclusion

Code was written for the CLAS12 validation suite for producing plots for the CTOF, FTOF, and forward and central vertex trackers. The reconstructed vertices of central and forward vertex trackers were plotted for un-extended and extended target cases. In the un-extended case, the vertex plots showed the resolution and precision for both detector sub-systems and provided resolution and precision measurements for each of the six FTOF detector sub-sectors. The extended vertex case was used compared improvements in the vertex precision. The improvement made in this thesis can be used to improve the vertex revolution, and momentum resolution. This paper also looked at the TOF resolution, and used this to identify particles in the FTOF, as well as how the improving the reconstructed velocity beta, by accounting for the vertex location can be used to improve TOF measurements and therefore increase particle resolution.

References

1. Gyurgyan , V, and Et Al. “CLARA: CLAS12 Reconstruction and Analysis Framework.” *Journal of Physics*, 1 Jan. 2016. *762 012009*.
2. Ungaro, M. “Background Merging Mechanism in GEMC .” 2018.
3. Baturin, V, et al. “TOF Resolution Measurements with the CLAS12 Central TOF Detector with Fine-Mesh Photomultiplier Tubes.” 1 Apr. 2011.
4. Burkert, V, et al. “Charged Particle Identification in Electron Scattering Experiments with CLAS.” 2 May 2002.
5. Carman, D S. “CLAS12 Central Time-of-Flight System Monte Carlo Simulation Details.” 12 Apr. 2016.
6. Carman, D S. “Forward Time-of-Flight Geometry for CLAS12.” 13 Apr. 2016.
7. Carman, D S. “CLAS12 Forward Time-of-Flight System Monte Carlo Simulation Details.” 12 Apr. 2016.
8. Carman, D S. “CLAS12 Time-of-Flight Systems Monte Carlo Simulation Details.” 6 Mar. 2016.
9. Carman, D S. “Forward Time-of-Flight Reconstruction for CLAS12.” 17 July 2017.
10. Carman, D S, et al. “Description of the Calibration Algorithms for the CLAS12 Central Time-Of-Flight System.” 19 Aug. 2018.
11. Carman, D S. “CLAS12 – Forward Time-of-Flight (FTOF) .” 8 Mar. 2018.

12. *CLAS12 Technical Design Report*. 5th ed., vol. 1 1, ser. 1, Thomas Jefferson National Accelerator Facility, 2008, *CLAS12 Technical Design Report*.

13. Gilman, R, and C Glashausser. "A Detailed Study of Semi-Inclusive Deep-Inelastic Pion Productions on Unpolerized Proton and Deuteron Targets with the CLAS12 Detector." A Letter-Of-Intent to Jefferson Lab PAC32

14. Gyurjyan, V, and D Abbott. "CLARA: A Contemporary Approach to Physics Data Processing." *Journal of Physics*, 1 Mar. 2013. *331 032013*.

15. Mecking, B A, and G Adams. "The CEBAF Large Acceptance Spectrometer (CLAS)." Nuclear Instruments and Methods in Physics Research , 2003, pp. 513–553.

16. Ziegler, V. "CLAS12 Reconstruction: Latest Improvements and Plans." CLAS12 Collaboration meeting . 18 June 2016, Newport News, Thomas Jefferson National Accelerator Facility.

17. Carman, D S. Work List 01312019. Work List 01312019.

18. "12 GeV Upgrade Technical Scope." *12 GeV Upgrade Technical Scope | Jefferson Lab*, www.jlab.org/physics/GeV.

19. Edwards, Kim. "JLAB History Archives Project." *Jefferson Lab Information Resources*, www.jlab.org/ir/archives/photos/weekly/2018/4-23-2018.html.

20. Sharabian, Y G, and L Elouadrhiri. “CLAS12 – High Threshold Cerenkov Counter (HTCC) .” 15 Mar. 2017.

21. staff, Science X. “Jefferson Lab Accelerator Delivers Its First 12 GeV Electrons.” *Phys.org*, Phys.org, 22 Dec. 2015, phys.org/news/2015-12-jefferson-lab-gev-electrons.html.

22. Ungaro, M, and L Elouadrhiri. “CLAS12 – Low Threshold Cerenkov Counter (LTCC).” 21 Mar. 2017.

23. Webmaster. “CLAS12.” *Jefferson Lab Experimental Hall B*, www.jlab.org/Hall-B/clas12-web/.

24. Baillie, Nathan Kidd. “Electron Scattering from an Almost Free Neutron in Deuterium .” *College of William and Mary*, 2010.

25. De Vita, Raffaella. “TrackingTest.java.” Git Hub, github.com/JeffersonLab/clas12-validation/blob/master/release-validation-1/src/main/java/org/jlab/c12val/TrackingTest.java.

Appendix A. Tracking Test Code.

Please see reference [25] for information about the source.

```
package trk;

//This imports java libraries, constants, data banks, and plot format.
import java.io.File;
import java.util.ArrayList;
import javax.swing.JFrame;
import org.jlab.clas.physics.LorentzVector;
import org.jlab.rec.eb.EBCCDBEnum;
import org.jlab.rec.eb.EBCCDBConstants;
import org.jlab.clas.pdg.PhysicsConstants;

import org.jlab.io.base.DataBank;
import org.jlab.io.base.DataEvent;
import org.jlab.io.hipo.HipoDataSource;

import org.jlab.groot.data.H1F;
import org.jlab.groot.data.H2F;
import org.jlab.groot.fitter.DataFitter;
import org.jlab.groot.math.F1D;

import org.jlab.clas.physics.Particle;
import org.jlab.groot.base.GStyle;
import org.jlab.groot.data.GraphErrors;
import org.jlab.groot.graphics.EmbeddedCanvasTabbed;
import org.jlab.groot.group.DataGroup;
import org.jlab.utils.groups.IndexedList;
```

```
/**
 *
 * Analyze tracking results
 *
 *
 *
 * @author devita and oliver
 */

public class TrackingTest {

   static final boolean debug=false;

   String  analysisName = "TrackingTest";

   IndexedList<DataGroup> dataGroups      = new IndexedList<DataGroup>(1);
   EmbeddedCanvasTabbed   canvasTabbed    = null;
   ArrayList<String>      canvasTabNames  = new ArrayList<String>();

   // these correspond to Joseph's two-particle event generater:
   static final int electronSector=1;
   static final int hadronSector=3;

   boolean isForwardTagger=false;
   boolean isCentral=false;

   int fdCharge = 0;

   int nNegTrackEvents = 0;
```

```
    int nTwoTrackEvents = 0;

    int nEvents = 0;

    double ebeam = 10.6;

    int particle_count=0;
    int event_count=0;

    float RF_Bucket_Length= 2.004f;
    float deltatr=0.0f;
    float rfCorr=0.0f;
    float Start_Time=0.0f;
    float Start_Time_Corr=0.0f;
    float vt_el=0.0f;

    float C = 29.9792458f;
    float Beta_=0.0f;

    //    @Test
    public static void main(String arg[]){
        //        System.setProperty("java.awt.headless", "true"); // this should disable the Xwindow
requirement
        GStyle.getAxisAttributesX().setTitleFontSize(24);
        GStyle.getAxisAttributesX().setLabelFontSize(18);
        GStyle.getAxisAttributesY().setTitleFontSize(24);
        GStyle.getAxisAttributesY().setLabelFontSize(18);
        GStyle.getAxisAttributesZ().setLabelFontSize(14);
        //        GStyle.setPalette("kDefault");
```

```
GStyle.getAxisAttributesX().setLabelFontName("Avenir");
GStyle.getAxisAttributesY().setLabelFontName("Avenir");
GStyle.getAxisAttributesZ().setLabelFontName("Avenir");
GStyle.getAxisAttributesX().setTitleFontName("Avenir");
GStyle.getAxisAttributesY().setTitleFontName("Avenir");
GStyle.getAxisAttributesZ().setTitleFontName("Avenir");
GStyle.setGraphicsFrameLineWidth(1);
GStyle.getH1FAttributes().setLineWidth(1);
GStyle.getH1FAttributes().setOptStat("1111");

TrackingTest ttest = new TrackingTest();


ttest.setAnalysisTabNames("Monte Carlo","Monte Carlo p negative", "Vertex","Positive
Tracks","Negative Tracks","CVT Positive Tracks","CVT Negative Tracks","CVT Vertex","Positive Tracks by
Sector","Negative Tracks by Sector","Vertex Comparison","Positive Tracks by Sector Main","Negative
Tracks by Sector Main","beta","Calculated Beta");

ttest.createHistos();

String resultDir=System.getProperty("RESULTS");
File dir = new File(resultDir);
if (!dir.isDirectory()) {
   System.err.println("Cannot find output directory");
   //assertEquals(false, true);
}
String inname = System.getProperty("INPUTFILE");
String fileName=resultDir + "/out_" + inname + ".hipo";
File file = new File(fileName);
if (!file.exists() || file.isDirectory()) {
```

```
        System.err.println("Cannot find input file.");
        //assertEquals(false, true); // use method from org.junit.Assert.* library to be able to do quantitative checks and test specific conditions
    }

    HipoDataSource reader = new HipoDataSource();
    reader.open(fileName);

    while (reader.hasEvent()) {
            DataEvent event = reader.getNextEvent();
      ttest.processEvent(event);
    }
    reader.close();

    JFrame frame = new JFrame("Tracking");
    frame.setSize(1200, 800);
    frame.add(ttest.canvasTabbed);
    frame.setLocationRelativeTo(null);
    frame.setVisible(true);

    ttest.analyze();
    ttest.plotHistos();

    ttest.printCanvas(resultDir,inname);
  }

  //processEvent() calls data banks from DataEvent and performs seletion criteria for producing the plots.
```

```
private void processEvent(DataEvent event) {
    nEvents++;
    if((nEvents%10000) == 0) System.out.println("Analyzed " + nEvents + " events");
    int run = 0;
    if(event.hasBank("RUN::config")) {
       run = event.getBank("RUN::config").getInt("run", 0);
    }
    else {
       return;
    }
    if(run>2365 && run<=2597)      ebeam=2.217;
    else if(run>3028 && run<=3105) ebeam=6.424;
    else if(run>3105 && run<=3817) ebeam=10.594;
    else if(run>3817 && run<=3861) ebeam=6.424;
    else if(run>3861)              ebeam=10.594;
    // process event info and save into data group
    Particle partGenNeg = null;
    Particle partGenPos = null;
    Particle partRecNeg = null;
    Particle partRecPos = null;
```

//This is the vertex comparison portion of the code. The following routine is for positive particle comparison. The code checks whether the event contains tracks detected by the forward and central detectors before executing the routine.

```
    if(event.hasBank("CVTRec::Tracks")==true && event.hasBank("TimeBasedTrkg::TBTracks")==true){
        DataBank  CVTbank = event.getBank("CVTRec::Tracks");
        DataBank  TBTbank = event.getBank("TimeBasedTrkg::TBTracks");
        int rowsCVT = CVTbank.rows();
```

```
DataBank  bank = event.getBank("CVTRec::Tracks");
for(int loop = 0; loop < rowsCVT; loop++){
   int pidCode=0;
   if(CVTbank.getByte("q", loop)==-1) pidCode = -211;
   else if(CVTbank.getByte("q", loop)==1) pidCode = 211;
   else pidCode = 22;
   Particle recParticleCVT = new Particle(
       pidCode,
       CVTbank.getFloat("pt", loop)*Math.cos(CVTbank.getFloat("phi0", loop)),
       CVTbank.getFloat("pt", loop)*Math.sin(CVTbank.getFloat("phi0", loop)),
       CVTbank.getFloat("pt", loop)*CVTbank.getFloat("tandip", loop),
       -CVTbank.getFloat("d0", loop)*Math.sin(CVTbank.getFloat("phi0", loop)),
       CVTbank.getFloat("d0", loop)*Math.cos(CVTbank.getFloat("phi0", loop)),
       CVTbank.getFloat("z0", loop));
   if(recParticleCVT.charge()>0) {

      int rowsTBT = TBTbank.rows();
      for(int loopTBT = 0; loopTBT < rowsTBT; loopTBT++){
         int pidCodeTBT=0;
         if(TBTbank.getByte("q", loopTBT)==-1) pidCodeTBT = 11;
         else if(TBTbank.getByte("q", loopTBT)==1) pidCodeTBT = 211;
         else pidCodeTBT = 22;
         Particle recParticleTBT = new Particle(
            pidCodeTBT,
            TBTbank.getFloat("p0_x", loopTBT),
            TBTbank.getFloat("p0_y", loopTBT),
            TBTbank.getFloat("p0_z", loopTBT),
            TBTbank.getFloat("Vtx0_x", loopTBT),
            TBTbank.getFloat("Vtx0_y", loopTBT),
```

```
                    TBTbank.getFloat("Vtx0_z", loopTBT));
                if(recParticleTBT.charge()>0) {
                    dataGroups.getItem(10).getH2F("vertex_CVT_TBT_pos").fill(recParticleTBT.vz(),
recParticleCVT.vz());
                    dataGroups.getItem(10).getH1F("vertex_CVT_TBT_diff_pos").fill(recParticleTBT.vz()-
recParticleCVT.vz());
                }
                if(partRecNeg==null && recParticleTBT.charge()<0) {
                    partRecNeg = new Particle();
                    partRecNeg.copy(recParticleTBT);
                }
                if(partRecPos==null && recParticleTBT.charge()>0) {
                    partRecPos = new Particle();
                    partRecPos.copy(recParticleTBT);
                }
              }
            }
            if(partRecNeg==null && recParticleCVT.charge()<0) {
              partRecNeg = new Particle();
              partRecNeg.copy(recParticleCVT);
            }
            if(partRecPos==null && recParticleCVT.charge()>0) {
              partRecPos = new Particle();
              partRecPos.copy(recParticleCVT);
            }
          }
    }

    //The following routine is for negative particle comparison.
```

```
if(event.hasBank("CVTRec::Tracks")==true && event.hasBank("TimeBasedTrkg::TBTracks")==true){
   DataBank  CVTbank = event.getBank("CVTRec::Tracks");
   DataBank  TBTbank = event.getBank("TimeBasedTrkg::TBTracks");
   int rowsCVT = CVTbank.rows();
   for(int loop = 0; loop < rowsCVT; loop++){
      int pidCode=0;
      if(CVTbank.getByte("q", loop)==-1) pidCode = -211;
      else if(CVTbank.getByte("q", loop)==1) pidCode = 211;
      else pidCode = 22;
      Particle recParticleCVT = new Particle(
          pidCode,
          CVTbank.getFloat("pt", loop)*Math.cos(CVTbank.getFloat("phi0", loop)),
          CVTbank.getFloat("pt", loop)*Math.sin(CVTbank.getFloat("phi0", loop)),
          CVTbank.getFloat("pt", loop)*CVTbank.getFloat("tandip", loop),
          -CVTbank.getFloat("d0", loop)*Math.sin(CVTbank.getFloat("phi0", loop)),
          CVTbank.getFloat("d0", loop)*Math.cos(CVTbank.getFloat("phi0", loop)),
          CVTbank.getFloat("z0", loop));
      if(recParticleCVT.charge()<0) {
         int rowsTBT = TBTbank.rows();
         for(int loopTBT = 0; loopTBT < rowsTBT; loopTBT++){
            int pidCodeTBT=0;
            if(TBTbank.getByte("q", loopTBT)==-1) pidCodeTBT = 11;
            else if(TBTbank.getByte("q", loopTBT)==1) pidCodeTBT = 211;
            else pidCodeTBT = 22;
            Particle recParticleTBT = new Particle(
               pidCodeTBT,
               TBTbank.getFloat("p0_x", loopTBT),
               TBTbank.getFloat("p0_y", loopTBT),
```

```
                TBTbank.getFloat("p0_z", loopTBT),
                TBTbank.getFloat("Vtx0_x", loopTBT),
                TBTbank.getFloat("Vtx0_y", loopTBT),
                TBTbank.getFloat("Vtx0_z", loopTBT));
            if(recParticleTBT.charge()<0) {
                dataGroups.getItem(10).getH2F("vertex_CVT_TBT_neg").fill(recParticleTBT.vz(),
recParticleCVT.vz());
                dataGroups.getItem(10).getH1F("vertex_CVT_TBT_diff_neg").fill(recParticleTBT.vz()-
recParticleCVT.vz());
            }
            if(partRecNeg==null && recParticleTBT.charge()<0) {
                partRecNeg = new Particle();
                partRecNeg.copy(recParticleTBT);
            }
            if(partRecPos==null && recParticleTBT.charge()>0) {
                partRecPos = new Particle();
                partRecPos.copy(recParticleTBT);
            }
          }
        }
        if(partRecNeg==null && recParticleCVT.charge()<0) {
          partRecNeg = new Particle();
          partRecNeg.copy(recParticleCVT);
        }
        if(partRecPos==null && recParticleCVT.charge()>0) {
          partRecPos = new Particle();
          partRecPos.copy(recParticleCVT);
        }
      }
```

```
}

//This portion plots the momentum vs the particle's velocity (beta). The velocity in this case comes from the reconstruction, and account for particles detected by the forward detector and central detectors.

if(event.hasBank("REC::Particle")==true && event.hasBank("REC::Event")==true){
   DataBank  recBank = event.getBank("REC::Particle");
   DataBank  evtBank = event.getBank("REC::Event");
   int recrows = recBank.rows();
   if(evtBank.getFloat("startTime",0)>-1000){
      if(recBank.getInt("pid",0)==11 && Math.abs(recBank.getShort("status",0))>2000 && Math.abs(recBank.getShort("status",0))<3000){
         event_count=event_count+1;
         for(int loop = 1; loop < recrows; loop++){
            int pidCode=0;
            if(recBank.getInt("charge", loop)==-1) pidCode = 11;
            else if(recBank.getInt("charge", loop)==1) pidCode = 211;
            else pidCode = 22;
            Particle newRecParticle = new Particle(
              pidCode,
              recBank.getFloat("px", loop),
              recBank.getFloat("py", loop),
              recBank.getFloat("pz", loop));
            newRecParticle.setProperty("beta", recBank.getFloat("beta", loop)); // method to attach "properties" to a barticle; the string can be user selected
            if(newRecParticle.charge()<0) {
               if(recBank.getShort("status", loop)>2000 && recBank.getShort("status", loop)<3000){

dataGroups.getItem(13).getH2F("p_vs_beta_neg").fill(newRecParticle.p(),newRecParticle.getProperty("beta"));
```

```
dataGroups.getItem(13).getH1F("neg_particle_count").fill(newRecParticle.getProperty("beta"));
                    }
                    if(recBank.getShort("status", loop)>4000){
dataGroups.getItem(13).getH2F("p_vs_beta_cvt_neg").fill(newRecParticle.p(),newRecParticle.getProperty("beta"));
dataGroups.getItem(13).getH1F("neg_particle_cvt_count").fill(newRecParticle.getProperty("beta"));
                    }
                }
                else {
                    if(recBank.getShort("status", loop)>2000 && recBank.getShort("status", loop)<3000){
dataGroups.getItem(13).getH2F("p_vs_beta_pos").fill(newRecParticle.p(),newRecParticle.getProperty("beta"));
dataGroups.getItem(13).getH1F("pos_particle_count").fill(newRecParticle.getProperty("beta"));
                    }
                    if(recBank.getShort("status", loop)>4000){
dataGroups.getItem(13).getH2F("p_vs_beta_cvt_pos").fill(newRecParticle.p(),newRecParticle.getProperty("beta"));
dataGroups.getItem(13).getH1F("pos_particle_cvt_count").fill(newRecParticle.getProperty("beta"));
                    }
                }
            }
        }
    }
```

//This portion also plots the momentum vs the particle's velocity (beta). In this case the velocity is calculated from scratch, and the vertex correction is added to the velocity calculation.

```
if(event.hasBank("REC::Particle")==true && event.hasBank("REC::Event")==true &&
event.hasBank("REC::Scintillator")==true){
    DataBank  recBank = event.getBank("REC::Particle");
    DataBank  evtBank = event.getBank("REC::Event");
    DataBank  sctBank = event.getBank("REC::Scintillator");
    int recrows = sctBank.rows();
    if(evtBank.getFloat("startTime",0)>-1000){
      if(recBank.getInt("pid",0)==11 && Math.abs(recBank.getShort("status",0))>2000 &&
Math.abs(recBank.getShort("status",0))<3000){
        event_count=event_count+1;
        for(int loop = 0; loop < recrows; loop++){
          if(sctBank.getShort("pindex", loop)==0 && sctBank.getByte("layer", loop)==2){
            vt_el= sctBank.getFloat("time", loop)-sctBank.getFloat("path",loop)/C;
            deltatr= -vt_el + evtBank.getFloat("RFTime",0) +1000.5f*RF_Bucket_Length;
            rfCorr = deltatr % RF_Bucket_Length - RF_Bucket_Length/2;
            Start_Time = vt_el + rfCorr;
          })
        }
        int recrows_2 = sctBank.rows();
        for(int loop = 0; loop < recrows_2; loop++){
          Beta_ = sctBank.getFloat("path",loop)/(sctBank.getFloat("time", loop)-Start_Time)/C;
          if(sctBank.getShort("pindex", loop)!=0 && sctBank.getByte("layer", loop)==2) {
            int index =sctBank.getShort("pindex",loop);
            int pidCode=0;
            if(recBank.getInt("charge", index)==-1) pidCode = -211;
            else if(recBank.getInt("charge", index)==1) pidCode = 211;
            else pidCode = 22;
```

```
                Particle newRecParticle_2 = new Particle(
                    pidCode,
                    recBank.getFloat("px", index),
                    recBank.getFloat("py", index),
                    recBank.getFloat("pz", index));
                float vz_corr=-recBank.getFloat("vz", index)/C;
                float vz_corr=0;
                deltatr= -vt_el + evtBank.getFloat("RFTime",0)-vz_corr + 1000.5f*RF_Bucket_Length;
                rfCorr = deltatr % RF_Bucket_Length - RF_Bucket_Length/2;
                Start_Time_Corr = vt_el + rfCorr;
                double betapi =
newRecParticle_2.p()/Math.sqrt(newRecParticle_2.p()*newRecParticle_2.p()+newRecParticle_2.mass2()
);
                double vtime     = sctBank.getFloat("time", loop)-sctBank.getFloat("path",loop)/C/betapi-
Start_Time;
                double vtimeCorr = sctBank.getFloat("time", loop)-
sctBank.getFloat("path",loop)/C/betapi-Start_Time_Corr;
                if(newRecParticle_2.charge()<0) {
                   dataGroups.getItem(14).getH2F("p_vs_Beta_neg").fill(newRecParticle_2.p(),Beta_);
                   dataGroups.getItem(14).getH1F("v_time_neg").fill(vtime);
                   dataGroups.getItem(14).getH1F("v_time_corr_neg").fill(vtimeCorr);
                   dataGroups.getItem(14).getH1F("Start_Time_Corr_neg").fill(deltatr);
                }
                if(newRecParticle_2.charge()>0) {
                   dataGroups.getItem(14).getH2F("p_vs_Beta_pos").fill(newRecParticle_2.p(),Beta_);
                }
              }
            }
          }
        }
```

```
    }

    //This portion is used for plots of the vertex reconstruction with respect to the forward detector.

    if(event.hasBank("TimeBasedTrkg::TBTracks")==true){
       DataBank  bank = event.getBank("TimeBasedTrkg::TBTracks");
       int rows = bank.rows();
       for(int loop = 0; loop < rows; loop++){
          int pidCode=0;
          if(bank.getByte("q", loop)==-1) pidCode = 11;
          else if(bank.getByte("q", loop)==1) pidCode = 211;
          else pidCode = 22;
          Particle recParticle = new Particle(
               pidCode,
               bank.getFloat("p0_x", loop),
               bank.getFloat("p0_y", loop),
               bank.getFloat("p0_z", loop),
               bank.getFloat("Vtx0_x", loop),
               bank.getFloat("Vtx0_y", loop),
               bank.getFloat("Vtx0_z", loop));
          if(bank.getShort("ndf", loop)>0) recParticle.setProperty("chi2", bank.getFloat("chi2",
loop)/bank.getShort("ndf", loop));
          if(recParticle.charge()>0) {
             dataGroups.getItem(2).getH1F("hi_p_pos").fill(recParticle.p());
             dataGroups.getItem(2).getH1F("hi_theta_pos").fill(Math.toDegrees(recParticle.theta()));
             dataGroups.getItem(2).getH1F("hi_phi_pos").fill(Math.toDegrees(recParticle.phi()));
             dataGroups.getItem(2).getH1F("hi_chi2_pos").fill(recParticle.getProperty("chi2"));
             dataGroups.getItem(2).getH1F("hi_vz_pos").fill(recParticle.vz());
```

```
dataGroups.getItem(4).getH2F("hi_vz_vs_theta_pos").fill(Math.toDegrees(recParticle.theta()),recParticle.vz());
            if(recParticle.p()>2.&& Math.toDegrees(recParticle.theta())>10.) {
               dataGroups.getItem(2).getH1F("hi_vz_pos_cut").fill(recParticle.vz());
               dataGroups.getItem(4).getH2F("hi_vxy_pos").fill(recParticle.vx(),recParticle.vy());
            }

dataGroups.getItem(2).getH2F("hi_theta_p_pos").fill(recParticle.p(),Math.toDegrees(recParticle.theta()));

dataGroups.getItem(2).getH2F("hi_theta_phi_pos").fill(Math.toDegrees(recParticle.phi()),Math.toDegrees(recParticle.theta()));

dataGroups.getItem(2).getH2F("hi_chi2_vz_pos").fill(recParticle.vz(),recParticle.getProperty("chi2"));
         }
         else {
            dataGroups.getItem(1).getH1F("hi_p_neg").fill(recParticle.p());
            dataGroups.getItem(1).getH1F("hi_theta_neg").fill(Math.toDegrees(recParticle.theta()));
            dataGroups.getItem(1).getH1F("hi_phi_neg").fill(Math.toDegrees(recParticle.phi()));
            dataGroups.getItem(1).getH1F("hi_chi2_neg").fill(recParticle.getProperty("chi2"));
            dataGroups.getItem(1).getH1F("hi_vz_neg").fill(recParticle.vz());

dataGroups.getItem(4).getH2F("hi_vz_vs_theta_neg").fill(Math.toDegrees(recParticle.theta()),recParticle.vz());
            if(recParticle.p()>2.&& Math.toDegrees(recParticle.theta())>15.) {
               dataGroups.getItem(1).getH1F("hi_vz_neg_cut").fill(recParticle.vz());
               dataGroups.getItem(4).getH2F("hi_vxy_neg").fill(recParticle.vx(),recParticle.vy());
            }

dataGroups.getItem(1).getH2F("hi_theta_p_neg").fill(recParticle.p(),Math.toDegrees(recParticle.theta()));
```

```
dataGroups.getItem(1).getH2F("hi_theta_phi_neg").fill(Math.toDegrees(recParticle.phi()),Math.toDegrees(recParticle.theta()));

dataGroups.getItem(1).getH2F("hi_chi2_vz_neg").fill(recParticle.vz(),recParticle.getProperty("chi2"));;
        }

        if (bank.getByte("sector", loop) == 1) {
           if(recParticle.charge()>0) {
              dataGroups.getItem(8).getH1F("hi_vz_pos_sec_1").fill(recParticle.vz());
              if(recParticle.p()>2.&& Math.toDegrees(recParticle.theta())>10.) {
                 dataGroups.getItem(8).getH1F("hi_vz_pos_sec_1_cut").fill(recParticle.vz());
              }
           }
           else {
              dataGroups.getItem(9).getH1F("hi_vz_neg_sec_1").fill(recParticle.vz());
              if(recParticle.p()>2.&& Math.toDegrees(recParticle.theta())>15.) {
                 dataGroups.getItem(9).getH1F("hi_vz_neg_sec_1_cut").fill(recParticle.vz());
              }
           }
        }
        if (bank.getByte("sector", loop) == 2) {
           if(recParticle.charge()>0) {
              dataGroups.getItem(8).getH1F("hi_vz_pos_sec_2").fill(recParticle.vz());
              if(recParticle.p()>2.&& Math.toDegrees(recParticle.theta())>10.) {
                 dataGroups.getItem(8).getH1F("hi_vz_pos_sec_2_cut").fill(recParticle.vz());
              }
           }
           else {
              dataGroups.getItem(9).getH1F("hi_vz_neg_sec_2").fill(recParticle.vz());
```

```
        if(recParticle.p()>2.&& Math.toDegrees(recParticle.theta())>15.) {
            dataGroups.getItem(9).getH1F("hi_vz_neg_sec_2_cut").fill(recParticle.vz());
        }
    }
}
if (bank.getByte("sector", loop) == 3) {
    if(recParticle.charge()>0) {
        dataGroups.getItem(8).getH1F("hi_vz_pos_sec_3").fill(recParticle.vz());
        if(recParticle.p()>2.&& Math.toDegrees(recParticle.theta())>10.) {
            dataGroups.getItem(8).getH1F("hi_vz_pos_sec_3_cut").fill(recParticle.vz());
        }
    }
    else {
        dataGroups.getItem(9).getH1F("hi_vz_neg_sec_3").fill(recParticle.vz());
        if(recParticle.p()>2.&& Math.toDegrees(recParticle.theta())>15.) {
            dataGroups.getItem(9).getH1F("hi_vz_neg_sec_3_cut").fill(recParticle.vz());
        }
    }
}
if (bank.getByte("sector", loop) == 4) {
    if(recParticle.charge()>0) {
        dataGroups.getItem(8).getH1F("hi_vz_pos_sec_4").fill(recParticle.vz());
        if(recParticle.p()>2.&& Math.toDegrees(recParticle.theta())>10.) {
            dataGroups.getItem(8).getH1F("hi_vz_pos_sec_4_cut").fill(recParticle.vz());
        }
    }
    else {
        dataGroups.getItem(9).getH1F("hi_vz_neg_sec_4").fill(recParticle.vz());
        if(recParticle.p()>2.&& Math.toDegrees(recParticle.theta())>15.) {
```

```
            dataGroups.getItem(9).getH1F("hi_vz_neg_sec_4_cut").fill(recParticle.vz());
        }
    }
}
if (bank.getByte("sector", loop) == 5) {
    if(recParticle.charge()>0) {
        dataGroups.getItem(8).getH1F("hi_vz_pos_sec_5").fill(recParticle.vz());
        if(recParticle.p()>2.&& Math.toDegrees(recParticle.theta())>10.) {
            dataGroups.getItem(8).getH1F("hi_vz_pos_sec_5_cut").fill(recParticle.vz());
        }
    }
    else {
        dataGroups.getItem(9).getH1F("hi_vz_neg_sec_5").fill(recParticle.vz());
        if(recParticle.p()>2.&& Math.toDegrees(recParticle.theta())>15.) {
            dataGroups.getItem(9).getH1F("hi_vz_neg_sec_5_cut").fill(recParticle.vz());
        }
    }
}
if (bank.getByte("sector", loop) == 6) {
    if(recParticle.charge()>0) {
        dataGroups.getItem(8).getH1F("hi_vz_pos_sec_6").fill(recParticle.vz());
        if(recParticle.p()>2.&& Math.toDegrees(recParticle.theta())>10.) {
            dataGroups.getItem(8).getH1F("hi_vz_pos_sec_6_cut").fill(recParticle.vz());
        }
    }
    else {
        dataGroups.getItem(9).getH1F("hi_vz_neg_sec_6").fill(recParticle.vz());
        if(recParticle.p()>2.&& Math.toDegrees(recParticle.theta())>15.) {
            dataGroups.getItem(9).getH1F("hi_vz_neg_sec_6_cut").fill(recParticle.vz());
```

```
            }
          }
        }
        if(partRecNeg==null && recParticle.charge()<0) {
          partRecNeg = new Particle();
          partRecNeg.copy(recParticle);
        }
        if(partRecPos==null && recParticle.charge()>0) {
          partRecPos = new Particle();
          partRecPos.copy(recParticle);
        }
      }
}

//This portion is the monte carlo comparison.

if(event.hasBank("MC::Particle")==true){
   DataBank genBank = event.getBank("MC::Particle");
   int nrows = genBank.rows();
   for(int loop = 0; loop < nrows; loop++) {
      Particle genPart = new Particle(
            genBank.getInt("pid", loop),
            genBank.getFloat("px", loop),
            genBank.getFloat("py", loop),
            genBank.getFloat("pz", loop),
            genBank.getFloat("vx", loop),
            genBank.getFloat("vy", loop),
            genBank.getFloat("vz", loop));
      if(genPart.pid()==11  && partGenNeg==null) partGenNeg=genPart;
```

```
            if(genPart.pid()==211 && partGenPos==null) partGenPos=genPart;

            if(genPart.charge()<0 && partGenNeg != null && partRecNeg != null) {

              if(testMCpart(genPart,partRecNeg)) {

                 dataGroups.getItem(3).getH1F("hi_dp_neg").fill((partRecNeg.p()-
partGenNeg.p())/partGenNeg.p());

                  dataGroups.getItem(3).getH2F("hi_dp_p_neg").fill(partGenNeg.p(),(partRecNeg.p()-
partGenNeg.p())/partGenNeg.p());


dataGroups.getItem(3).getH2F("hi_dp_theta_neg").fill(Math.toDegrees(partGenNeg.theta()),(partRecNe
g.p()-partGenNeg.p())/partGenNeg.p());


dataGroups.getItem(3).getH2F("hi_dp_phi_neg").fill(Math.toDegrees(partGenNeg.phi()),(partRecNeg.p()
-partGenNeg.p())/partGenNeg.p());

                  dataGroups.getItem(3).getH1F("hi_dtheta_neg").fill(Math.toDegrees(partRecNeg.theta()-
partGenNeg.theta()));

                  dataGroups.getItem(3).getH1F("hi_dphi_neg").fill(Math.toDegrees(partRecNeg.phi()-
partGenNeg.phi()));

                  dataGroups.getItem(3).getH1F("hi_dvz_neg").fill(partRecNeg.vz()-partGenNeg.vz());

              }

            }

            if(genPart.charge()>0 && partGenPos != null && partRecPos != null) {

              if(testMCpart(genPart,partRecPos)) {

                  dataGroups.getItem(3).getH1F("hi_dp_pos").fill((partRecPos.p()-
partGenPos.p())/partGenPos.p());

                  dataGroups.getItem(3).getH1F("hi_dtheta_pos").fill(Math.toDegrees(partRecPos.theta()-
partGenPos.theta()));

                  dataGroups.getItem(3).getH1F("hi_dphi_pos").fill(Math.toDegrees(partRecPos.phi()-
partGenPos.phi()));

                  dataGroups.getItem(3).getH1F("hi_dvz_pos").fill(partRecPos.vz()-partGenPos.vz());

              }

            }

          }

        }
```

```
//This portion is used for plots of the vertex reconstruction with respect to the central detector.


if(event.hasBank("CVTRec::Tracks")==true){
   DataBank  bank = event.getBank("CVTRec::Tracks");
   int rows = bank.rows();
   for(int loop = 0; loop < rows; loop++){
      int pidCode=0;
      if(bank.getByte("q", loop)==-1) pidCode = -211;
      else if(bank.getByte("q", loop)==1) pidCode = 211;
      else pidCode = 22;;
      Particle recParticle = new Particle(
          pidCode,
          bank.getFloat("pt", loop)*Math.cos(bank.getFloat("phi0", loop)),
          bank.getFloat("pt", loop)*Math.sin(bank.getFloat("phi0", loop)),
          bank.getFloat("pt", loop)*bank.getFloat("tandip", loop),
          -bank.getFloat("d0", loop)*Math.sin(bank.getFloat("phi0", loop)),
          bank.getFloat("d0", loop)*Math.cos(bank.getFloat("phi0", loop)),
          bank.getFloat("z0", loop));
      if(bank.getShort("ndf", loop)>0) recParticle.setProperty("chi2", bank.getFloat("chi2",
loop)/bank.getShort("ndf", loop));
      if(recParticle.charge()>0) {
         dataGroups.getItem(6).getH1F("hi_p_pos_cvt").fill(recParticle.p());
         dataGroups.getItem(6).getH1F("hi_theta_pos_cvt").fill((float)
Math.toDegrees(recParticle.theta()));
         dataGroups.getItem(6).getH1F("hi_phi_pos_cvt").fill(Math.toDegrees(recParticle.phi()));
         dataGroups.getItem(6).getH1F("hi_vz_pos_cvt").fill(recParticle.vz());

dataGroups.getItem(6).getH2F("hi_theta_p_pos_cvt").fill(recParticle.p(),Math.toDegrees(recParticle.the
ta()));
```

```
dataGroups.getItem(6).getH2F("hi_theta_phi_pos_cvt").fill(Math.toDegrees(recParticle.phi()),Math.toDegrees(recParticle.theta()));
            dataGroups.getItem(6).getH1F("hi_chi2_pos_cvt").fill(recParticle.getProperty("chi2"));

dataGroups.getItem(7).getH2F("hi_vz_vs_theta_pos_cvt").fill(Math.toDegrees(recParticle.theta()),recParticle.vz());
            if(recParticle.p()>2.&& Math.toDegrees(recParticle.theta())>10.) {
               dataGroups.getItem(6).getH1F("hi_vz_pos_cvt_cut").fill(recParticle.vz());
               dataGroups.getItem(7).getH2F("hi_vxy_pos_cvt").fill(recParticle.vx(),recParticle.vy());
            }
         }
         else {
            dataGroups.getItem(5).getH1F("hi_p_neg_cvt").fill(recParticle.p());
            dataGroups.getItem(5).getH1F("hi_theta_neg_cvt").fill(Math.toDegrees(recParticle.theta()));
            dataGroups.getItem(5).getH1F("hi_phi_neg_cvt").fill(Math.toDegrees(recParticle.phi()));
            dataGroups.getItem(5).getH1F("hi_vz_neg_cvt").fill(recParticle.vz());

dataGroups.getItem(5).getH2F("hi_theta_p_neg_cvt").fill(recParticle.p(),Math.toDegrees(recParticle.theta()));

dataGroups.getItem(5).getH2F("hi_theta_phi_neg_cvt").fill(Math.toDegrees(recParticle.phi()),Math.toDegrees(recParticle.theta()));
            dataGroups.getItem(5).getH1F("hi_chi2_neg_cvt").fill(recParticle.getProperty("chi2"));

dataGroups.getItem(7).getH2F("hi_vz_vs_theta_neg_cvt").fill(Math.toDegrees(recParticle.theta()),recParticle.vz());
            if(recParticle.p()>.5&& Math.toDegrees(recParticle.theta())>15.) {
               dataGroups.getItem(7).getH2F("hi_vxy_neg_cvt").fill(recParticle.vx(),recParticle.vy());
            }

dataGroups.getItem(5).getH2F("hi_chi2_vz_neg_cvt").fill(recParticle.vz(),recParticle.getProperty("chi2"));
```

```
        }
        if(partRecNeg==null && recParticle.charge()<0) {
           partRecNeg = new Particle();
           partRecNeg.copy(recParticle);
        }
        if(partRecPos==null && recParticle.charge()>0) {
           partRecPos = new Particle();
           partRecPos.copy(recParticle);
        }
      }
   }
}

//createHistos() creates histograms from the data in the data banks specified in processEvent

private void createHistos() {
   // negative tracks
   H1F hi_p_neg = new H1F("hi_p_neg", "hi_p_neg", 100, 0.0, 8.0);
   hi_p_neg.setTitleX("p (GeV)");
   hi_p_neg.setTitleY("Counts");
   H1F hi_theta_neg = new H1F("hi_theta_neg", "hi_theta_neg", 100, 0.0, 40.0);
   hi_theta_neg.setTitleX("#theta (deg)");
   hi_theta_neg.setTitleY("Counts");
   H1F hi_phi_neg = new H1F("hi_phi_neg", "hi_phi_neg", 100, -180.0, 180.0);
   hi_phi_neg.setTitleX("#phi (deg)");
   hi_phi_neg.setTitleY("Counts");
   H1F hi_chi2_neg = new H1F("hi_chi2_neg", "hi_chi2_neg", 100, 0.0, 180.0);
   hi_chi2_neg.setTitleX("#chi2");
   hi_chi2_neg.setTitleY("Counts");
```

```
H1F hi_vz_neg = new H1F("hi_vz_neg", "hi_vz_neg", 100, -15.0, 15.0);
hi_vz_neg.setTitleX("Vz (cm)");
hi_vz_neg.setTitleY("Counts");
H1F hi_vz_neg_cut = new H1F("hi_vz_neg_cut", "hi_vz_neg_cut", 100, -15.0, 15.0);
hi_vz_neg_cut.setTitleX("Vz (cm)");
hi_vz_neg_cut.setTitleY("Counts");
hi_vz_neg_cut.setLineColor(2);
F1D f1_vz_neg = new F1D("f1_vz_neg","[amp]*gaus(x,[mean],[sigma])", -5.0, 5.0);
f1_vz_neg.setParameter(0, 0);
f1_vz_neg.setParameter(1, 0);
f1_vz_neg.setParameter(2, 1.0);
f1_vz_neg.setLineWidth(2);
f1_vz_neg.setLineColor(1);
f1_vz_neg.setOptStat("1111");
H2F hi_theta_p_neg = new H2F("hi_theta_p_neg", "hi_theta_p_neg", 100, 0.0, 8.0, 100, 0.0, 40.0);
hi_theta_p_neg.setTitleX("p (GeV)");
hi_theta_p_neg.setTitleY("#theta (deg)");
H2F hi_theta_phi_neg = new H2F("hi_theta_phi_neg", "hi_theta_phi_neg", 100, -180.0, 180.0, 100,
0.0, 40.0);
hi_theta_phi_neg.setTitleX("#phi (deg)");
hi_theta_phi_neg.setTitleY("#theta (deg)");
H2F hi_chi2_vz_neg = new H2F("hi_chi2_vz_neg", "hi_chi2_vz_neg", 100, -15.0, 15.0, 100, 0.0,
180.0);
hi_chi2_vz_neg.setTitleX("Vz (cm)");
hi_chi2_vz_neg.setTitleY("#chi2");
DataGroup dg_neg = new DataGroup(4,2);
dg_neg.addDataSet(hi_p_neg, 0);
dg_neg.addDataSet(hi_theta_neg, 1);
dg_neg.addDataSet(hi_phi_neg, 2);
```

```
dg_neg.addDataSet(hi_chi2_neg, 3);
dg_neg.addDataSet(hi_vz_neg, 4);
dg_neg.addDataSet(hi_vz_neg_cut, 4);
dg_neg.addDataSet(f1_vz_neg, 4);
dg_neg.addDataSet(hi_theta_p_neg, 5);
dg_neg.addDataSet(hi_theta_phi_neg, 6);
dg_neg.addDataSet(hi_chi2_vz_neg, 7);
dataGroups.add(dg_neg, 1);
// positive trakcs
H1F hi_p_pos = new H1F("hi_p_pos", "hi_p_pos", 100, 0.0, 8.0);
hi_p_pos.setTitleX("p (GeV)");
hi_p_pos.setTitleY("Counts");
H1F hi_theta_pos = new H1F("hi_theta_pos", "hi_theta_pos", 100, 0.0, 40.0);
hi_theta_pos.setTitleX("#theta (deg)");
hi_theta_pos.setTitleY("Counts");
H1F hi_phi_pos = new H1F("hi_phi_pos", "hi_phi_pos", 100, -180.0, 180.0);
hi_phi_pos.setTitleX("#phi (deg)");
hi_phi_pos.setTitleY("Counts");
H1F hi_chi2_pos = new H1F("hi_chi2_pos", "hi_chi2_pos", 100, 0.0, 180.0);
hi_chi2_pos.setTitleX("#chi2");
hi_chi2_pos.setTitleY("Counts");
H1F hi_vz_pos = new H1F("hi_vz_pos", "hi_vz_pos", 100, -15.0, 15.0);
hi_vz_pos.setTitleX("Vz (cm)");
hi_vz_pos.setTitleY("Counts");
H1F hi_vz_pos_cut = new H1F("hi_vz_pos_cut", "hi_vz_pos_cut", 100, -15.0, 15.0);
hi_vz_pos_cut.setTitleX("Vz (cm)");
hi_vz_pos_cut.setTitleY("Counts");
hi_vz_pos_cut.setLineColor(2);
F1D f1_vz_pos = new F1D("f1_vz_pos","[amp]*gaus(x,[mean],[sigma])", -5.0, 5.0);
```

```
f1_vz_pos.setParameter(0, 0);
f1_vz_pos.setParameter(1, 0);
f1_vz_pos.setParameter(2, 1.0);
f1_vz_pos.setLineWidth(2);
f1_vz_pos.setLineColor(1);
f1_vz_pos.setOptStat("1111");
H2F hi_theta_p_pos = new H2F("hi_theta_p_pos", "hi_theta_p_pos", 100, 0.0, 8.0, 100, 0.0, 40.0);
hi_theta_p_pos.setTitleX("p (GeV)");
hi_theta_p_pos.setTitleY("#theta (deg)");
H2F hi_theta_phi_pos = new H2F("hi_theta_phi_pos", "hi_theta_phi_pos", 100, -180.0, 180.0, 100, 0.0, 40.0);
hi_theta_phi_pos.setTitleX("#phi (deg)");
hi_theta_phi_pos.setTitleY("#theta (deg)");
H2F hi_chi2_vz_pos = new H2F("hi_chi2_vz_pos", "hi_chi2_vz_pos", 100, -15.0, 15.0, 100, 0.0, 180.0);
hi_chi2_vz_pos.setTitleX("Vz (cm)");
hi_chi2_vz_pos.setTitleY("#chi2");
DataGroup dg_pos = new DataGroup(4,2);
dg_pos.addDataSet(hi_p_pos, 0);
dg_pos.addDataSet(hi_theta_pos, 1);
dg_pos.addDataSet(hi_phi_pos, 2);
dg_pos.addDataSet(hi_chi2_pos, 3);
dg_pos.addDataSet(hi_vz_pos, 4);
dg_pos.addDataSet(hi_vz_pos_cut, 4);
dg_pos.addDataSet(f1_vz_pos, 4);
dg_pos.addDataSet(hi_theta_p_pos, 5);
dg_pos.addDataSet(hi_theta_phi_pos, 6);
dg_pos.addDataSet(hi_chi2_vz_pos, 7);
dataGroups.add(dg_pos, 2);
```

```
// mc comparison
H1F hi_dp_pos = new H1F("hi_dp_pos", "hi_dp_pos", 100, -0.5, 0.5);
hi_dp_pos.setTitleX("#Delta P/P");
hi_dp_pos.setTitleY("Counts");
hi_dp_pos.setTitle("Positive Tracks");
F1D f1_dp_pos = new F1D("f1_dp_pos","[amp]*gaus(x,[mean],[sigma])", -0.5, 0.5);
f1_dp_pos.setParameter(0, 0);
f1_dp_pos.setParameter(1, 0);
f1_dp_pos.setParameter(2, 1.0);
f1_dp_pos.setLineWidth(2);
f1_dp_pos.setLineColor(2);
f1_dp_pos.setOptStat("1111");
H1F hi_dtheta_pos = new H1F("hi_dtheta_pos","hi_dtheta_pos", 100, -4.0, 4.0);
hi_dtheta_pos.setTitleX("#Delta #theta (deg)");
hi_dtheta_pos.setTitleY("Counts");
hi_dtheta_pos.setTitle("Positive Tracks");
F1D f1_dtheta_pos = new F1D("f1_dtheta_pos","[amp]*gaus(x,[mean],[sigma])", -10.0, 10.0);
f1_dtheta_pos.setParameter(0, 0);
f1_dtheta_pos.setParameter(1, 0);
f1_dtheta_pos.setParameter(2, 1.0);
f1_dtheta_pos.setLineWidth(2);
f1_dtheta_pos.setLineColor(2);
f1_dtheta_pos.setOptStat("1111");
H1F hi_dphi_pos = new H1F("hi_dphi_pos", "hi_dphi_pos", 100, -8.0, 8.0);
hi_dphi_pos.setTitleX("#Delta #phi (deg)");
hi_dphi_pos.setTitleY("Counts");
hi_dphi_pos.setTitle("Positive Tracks");
F1D f1_dphi_pos = new F1D("f1_dphi_pos","[amp]*gaus(x,[mean],[sigma])", -10.0, 10.0);
f1_dphi_pos.setParameter(0, 0);
```

```
f1_dphi_pos.setParameter(1, 0);
f1_dphi_pos.setParameter(2, 1.0);
f1_dphi_pos.setLineWidth(2);
f1_dphi_pos.setLineColor(2);
f1_dphi_pos.setOptStat("1111");
H1F hi_dvz_pos = new H1F("hi_dvz_pos", "hi_dvz_pos", 100, -20.0, 20.0);
hi_dvz_pos.setTitleX("#Delta Vz (cm)");
hi_dvz_pos.setTitleY("Counts");
hi_dvz_pos.setTitle("Positive Tracks");
F1D f1_dvz_pos = new F1D("f1_dvz_pos","[amp]*gaus(x,[mean],[sigma])", -10.0, 10.0);
f1_dvz_pos.setParameter(0, 0);
f1_dvz_pos.setParameter(1, 0);
f1_dvz_pos.setParameter(2, 1.0);
f1_dvz_pos.setLineWidth(2);
f1_dvz_pos.setLineColor(2);
f1_dvz_pos.setOptStat("1111");
H1F hi_dp_neg = new H1F("hi_dp_neg", "hi_dp_neg", 100, -0.5, 0.5);
hi_dp_neg.setTitleX("#Delta P/P");
hi_dp_neg.setTitleY("Counts");
hi_dp_neg.setTitle("Negative Tracks");
F1D f1_dp_neg = new F1D("f1_dp_neg","[amp]*gaus(x,[mean],[sigma])",  -0.5, 0.5);
f1_dp_neg.setParameter(0, 0);
f1_dp_neg.setParameter(1, 0);
f1_dp_neg.setParameter(2, 1.0);
f1_dp_neg.setLineWidth(2);
f1_dp_neg.setLineColor(2);
f1_dp_neg.setOptStat("1111");
H1F hi_dtheta_neg = new H1F("hi_dtheta_neg","hi_dtheta_neg", 100, -4.0, 4.0);
hi_dtheta_neg.setTitleX("#Delta #theta (deg)");
```

```
hi_dtheta_neg.setTitleY("Counts");
hi_dtheta_neg.setTitle("Negative Tracks");
F1D f1_dtheta_neg = new F1D("f1_dtheta_neg","[amp]*gaus(x,[mean],[sigma])", -10.0, 10.0);
f1_dtheta_neg.setParameter(0, 0);
f1_dtheta_neg.setParameter(1, 0);
f1_dtheta_neg.setParameter(2, 1.0);
f1_dtheta_neg.setLineWidth(2);
f1_dtheta_neg.setLineColor(2);
f1_dtheta_neg.setOptStat("1111");
H1F hi_dphi_neg = new H1F("hi_dphi_neg", "hi_dphi_neg", 100, -8.0, 8.0);
hi_dphi_neg.setTitleX("#Delta #phi (deg)");
hi_dphi_neg.setTitleY("Counts");
hi_dphi_neg.setTitle("Negative Tracks");
F1D f1_dphi_neg = new F1D("f1_dphi_neg","[amp]*gaus(x,[mean],[sigma])", -10.0, 10.0);
f1_dphi_neg.setParameter(0, 0);
f1_dphi_neg.setParameter(1, 0);
f1_dphi_neg.setParameter(2, 1.0);
f1_dphi_neg.setLineWidth(2);
f1_dphi_neg.setLineColor(2);
f1_dphi_neg.setOptStat("1111");
H1F hi_dvz_neg = new H1F("hi_dvz_neg", "hi_dvz_neg", 100, -20.0, 20.0);
hi_dvz_neg.setTitleX("#Delta Vz (cm)");
hi_dvz_neg.setTitleY("Counts");
hi_dvz_neg.setTitle("Negative Tracks");
F1D f1_dvz_neg = new F1D("f1_dvz_neg","[amp]*gaus(x,[mean],[sigma])", -10.0, 10.0);
f1_dvz_neg.setParameter(0, 0);
f1_dvz_neg.setParameter(1, 0);
f1_dvz_neg.setParameter(2, 1.0);
f1_dvz_neg.setLineWidth(2);
```

```
f1_dvz_neg.setLineColor(2);
f1_dvz_neg.setOptStat("1111");
H2F hi_dp_p_neg = new H2F("hi_dp_p_neg", "hi_dp_p_neg", 16, 0.5, 8.5, 100, -0.2, 0.2);
hi_dp_p_neg.setTitleX("p");
hi_dp_p_neg.setTitleY("#Delta p/p");
hi_dp_p_neg.setTitle("Negative Tracks");
H2F hi_dp_theta_neg = new H2F("hi_dp_theta_neg", "hi_dp_theta_neg", 30, 5, 35, 100, -0.2, 0.2);
hi_dp_theta_neg.setTitleX("#theta");
hi_dp_theta_neg.setTitleY("#Delta p/p");
hi_dp_theta_neg.setTitle("Negative Tracks");
H2F hi_dp_phi_neg = new H2F("hi_dp_phi_neg", "hi_dp_phi_neg", 100, -180, 180, 100, -0.2, 0.2);
hi_dp_phi_neg.setTitleX("#phi");
hi_dp_phi_neg.setTitleY("#Delta p/p");
hi_dp_phi_neg.setTitle("Negative Tracks");
GraphErrors gr_dp_p_neg = new GraphErrors("gr_dp_p_neg");
gr_dp_p_neg.setTitleX("p");
gr_dp_p_neg.setTitleY("#Delta p/p");
gr_dp_p_neg.setTitle("gr_dp_p_neg");
GraphErrors gr_dp_theta_neg = new GraphErrors("gr_dp_theta_neg");
gr_dp_theta_neg.setTitleX("#theta");
gr_dp_theta_neg.setTitleY("#Delta p/p");
gr_dp_theta_neg.setTitle("gr_dp_p_neg");
GraphErrors gr_dp_phi_neg = new GraphErrors("gr_dp_phi_neg");
gr_dp_phi_neg.setTitleX("#phi");
gr_dp_phi_neg.setTitleY("#Delta p/p");
gr_dp_phi_neg.setTitle("gr_dp_p_neg");
DataGroup mc = new DataGroup(4,2);
mc.addDataSet(hi_dp_pos, 0);
mc.addDataSet(f1_dp_pos, 0);
```

```
mc.addDataSet(hi_dtheta_pos, 1);
mc.addDataSet(f1_dtheta_pos, 1);
mc.addDataSet(hi_dphi_pos, 2);
mc.addDataSet(f1_dphi_pos, 2);
mc.addDataSet(hi_dvz_pos, 3);
mc.addDataSet(f1_dvz_pos, 3);
mc.addDataSet(hi_dp_neg, 4);
mc.addDataSet(f1_dp_neg, 4);
mc.addDataSet(hi_dtheta_neg, 5);
mc.addDataSet(f1_dtheta_neg, 5);
mc.addDataSet(hi_dphi_neg, 6);
mc.addDataSet(f1_dphi_neg, 6);
mc.addDataSet(hi_dvz_neg, 7);
mc.addDataSet(f1_dvz_neg, 7);
mc.addDataSet(hi_dp_p_neg, 8);
mc.addDataSet(hi_dp_theta_neg, 9);
mc.addDataSet(hi_dp_phi_neg, 10);
mc.addDataSet(gr_dp_p_neg, 11);
mc.addDataSet(gr_dp_theta_neg, 12);
mc.addDataSet(gr_dp_phi_neg, 13);
dataGroups.add(mc, 3);
// vertex
H2F hi_vxy_pos = new H2F("hi_vxy_pos","hi_vxy_pos",100,-15.,15.,100,-15.,15);
hi_vxy_pos.setTitleX("Vx (cm)");
hi_vxy_pos.setTitleY("Vy (cm)");
H2F hi_vxy_neg = new H2F("hi_vxy_neg","hi_vxy_neg",100,-15.,15.,100,-15.,15);
hi_vxy_neg.setTitleX("Vx (cm)");
hi_vxy_neg.setTitleY("Vy (cm)");
```

```
    H2F hi_vz_vs_theta_pos = new H2F("hi_vz_vs_theta_pos","hi_vz_vs_theta_pos",100, 5.,40.,100,-15.,15);
    hi_vz_vs_theta_pos.setTitleX("#theta (deg)");
    hi_vz_vs_theta_pos.setTitleY("Vz (cm)");
    H2F hi_vz_vs_theta_neg = new H2F("hi_vz_vs_theta_neg","hi_vz_vs_theta_neg",100, 5.,40.,100,-15.,15);
    hi_vz_vs_theta_neg.setTitleX("#theta (deg)");
    hi_vz_vs_theta_neg.setTitleY("Vz (cm)");
    DataGroup vertex = new DataGroup(2,2);
    vertex.addDataSet(hi_vz_vs_theta_pos, 0);
    vertex.addDataSet(hi_vxy_pos, 1);
    vertex.addDataSet(hi_vz_vs_theta_neg, 2);
    vertex.addDataSet(hi_vxy_neg, 3);
    dataGroups.add(vertex, 4);

    // CVT Negative Tracks
    H1F hi_p_neg_cvt = new H1F("hi_p_neg_cvt", "hi_p_neg_cvt", 100, 0.0, 3.0);
    hi_p_neg_cvt.setTitleX("p (GeV)");
    hi_p_neg_cvt.setTitleY("Counts");
    H1F hi_theta_neg_cvt = new H1F("hi_theta_neg_cvt", "hi_theta_neg_cvt", 100, 0.0, 120.0);
    hi_theta_neg_cvt.setTitleX("#theta (deg)");
    hi_theta_neg_cvt.setTitleY("Counts");
    H1F hi_phi_neg_cvt = new H1F("hi_phi_neg_cvt", "hi_phi_neg_cvt", 100, -180.0, 180.0);
    hi_phi_neg_cvt.setTitleX("#phi (deg)");
    H1F hi_chi2_neg_cvt = new H1F("hi_chi2_neg_cvt", "hi_chi2_neg_cvt", 100, 0.0, 180.0);
    hi_chi2_neg_cvt.setTitleX("#chi2");
    hi_chi2_neg_cvt.setTitleY("Counts");
    H1F hi_vz_neg_cvt = new H1F("hi_vz_neg_cvt", "hi_vz_neg_cvt", 100, -10.0, 10.0);
    hi_vz_neg_cvt.setTitleX("Vz (cm)");
```

```
hi_vz_neg_cvt.setTitleY("Counts");

H2F hi_theta_p_neg_cvt = new H2F("hi_theta_p_neg_cvt", "hi_theta_p_neg_cvt", 100, 0.0, 3.0, 100, 0.0, 120.0);

hi_theta_p_neg_cvt.setTitleX("p (GeV)");

hi_theta_p_neg_cvt.setTitleY("#theta (deg)");

H2F hi_theta_phi_neg_cvt = new H2F("hi_theta_phi_neg_cvt", "hi_theta_phi_neg_cvt", 100, -180.0, 180.0, 100, 0.0, 120.0);

hi_theta_phi_neg_cvt.setTitleX("#phi (deg)");

hi_theta_phi_neg_cvt.setTitleY("#theta (deg)");

F1D f1_vz_neg_cvt = new F1D("f1_vz_neg_cvt","[amp]*gaus(x,[mean],[sigma])", -5.0, 5.0);

f1_vz_neg_cvt.setParameter(0, 0);

f1_vz_neg_cvt.setParameter(1, 0);

f1_vz_neg_cvt.setParameter(2, 1.0);

f1_vz_neg_cvt.setLineWidth(2);

f1_vz_neg_cvt.setLineColor(1);

f1_vz_neg_cvt.setOptStat("1111");

H2F hi_chi2_vz_neg_cvt = new H2F("hi_chi2_vz_neg_cvt", "hi_chi2_vz_neg_cvt", 100, -15.0, 15.0, 100, 0.0, 180.0);

hi_chi2_vz_neg_cvt.setTitleX("Vz (cm)");

hi_chi2_vz_neg_cvt.setTitleY("#chi2");

DataGroup dg_neg_cvt = new DataGroup(4,2);

dg_neg_cvt.addDataSet(hi_p_neg_cvt, 0);

dg_neg_cvt.addDataSet(hi_theta_neg_cvt, 1);

dg_neg_cvt.addDataSet(hi_phi_neg_cvt, 2);

dg_neg_cvt.addDataSet(hi_chi2_neg_cvt, 3);

dg_neg_cvt.addDataSet(hi_vz_neg_cvt, 3);

dg_neg_cvt.addDataSet(f1_vz_neg_cvt, 4);

dg_neg_cvt.addDataSet(hi_theta_p_neg_cvt, 5);

dg_neg_cvt.addDataSet(hi_theta_phi_neg_cvt, 6);

dg_neg_cvt.addDataSet(hi_chi2_vz_neg_cvt, 7);
```

```
dataGroups.add(dg_neg_cvt, 5);
// cvt positive trakcs
H1F hi_p_pos_cvt = new H1F("hi_p_pos_cvt", "hi_p_pos_cvt", 100, 0.0, 3.0);
hi_p_pos_cvt.setTitleX("p (GeV)");
hi_p_pos_cvt.setTitleY("Counts");
H1F hi_theta_pos_cvt = new H1F("hi_theta_pos_cvt", "hi_theta_pos_cvt", 100, 0.0, 120.0);
hi_theta_pos_cvt.setTitleX("#theta (deg)");
hi_theta_pos_cvt.setTitleY("Counts");
H1F hi_phi_pos_cvt = new H1F("hi_phi_pos_cvt", "hi_phi_pos_cvt", 100, -180.0, 180.0);
hi_phi_pos_cvt.setTitleX("#phi (deg)");
H1F hi_chi2_pos_cvt = new H1F("hi_chi2_pos_cvt", "hi_chi2_pos_cvt", 100, 0.0, 180.0);
hi_chi2_pos_cvt.setTitleX("#chi2");
hi_chi2_pos_cvt.setTitleY("Counts");
H1F hi_vz_pos_cvt = new H1F("hi_vz_pos_cvt", "hi_vz_pos_cvt", 100, -10.0, 10.0);
hi_vz_pos_cvt.setTitleX("Vz (cm)");
hi_vz_pos_cvt.setTitleY("Counts");
H2F hi_theta_p_pos_cvt = new H2F("hi_theta_p_pos_cvt", "hi_theta_p_pos_cvt", 100, 0.0, 3.0, 100,
0.0, 120.0);
hi_theta_p_pos_cvt.setTitleX("p (GeV)");
hi_theta_p_pos_cvt.setTitleY("#theta (deg)");
H2F hi_theta_phi_pos_cvt = new H2F("hi_theta_phi_pos_cvt", "hi_theta_phi_pos_cvt", 100, -180.0,
180.0, 100, 0.0, 120.0);
hi_theta_phi_pos_cvt.setTitleX("#phi (deg)");
hi_theta_phi_pos_cvt.setTitleY("#theta (deg)");
H1F hi_vz_pos_cvt_cut = new H1F("hi_vz_pos_cvt_cut", "hi_vz_pos_cvt_cut", 100, -15.0, 15.0);
hi_vz_pos_cvt_cut.setTitleX("Vz (cm)");
hi_vz_pos_cvt_cut.setTitleY("Counts");
hi_vz_pos_cvt_cut.setLineColor(2);
F1D f1_vz_pos_cvt = new F1D("f1_vz_pos_cvt","[amp]*gaus(x,[mean],[sigma])", -5.0, 5.0);
```

```
f1_vz_pos_cvt.setParameter(0, 0);
f1_vz_pos_cvt.setParameter(1, 0);
f1_vz_pos_cvt.setParameter(2, 1.0);
f1_vz_pos_cvt.setLineWidth(2);
f1_vz_pos_cvt.setLineColor(1);
f1_vz_pos_cvt.setOptStat("1111");
DataGroup dg_pos_cvt = new DataGroup(4,4);
dg_pos_cvt.addDataSet(hi_p_pos_cvt, 0);
dg_pos_cvt.addDataSet(hi_theta_pos_cvt, 1);
dg_pos_cvt.addDataSet(hi_phi_pos_cvt, 2);
dg_pos_cvt.addDataSet(hi_chi2_pos_cvt, 3);
dg_pos_cvt.addDataSet(hi_vz_pos_cvt, 4);
dg_pos_cvt.addDataSet(hi_vz_pos_cvt_cut, 4);
dg_pos_cvt.addDataSet(f1_vz_pos_cvt, 4);
dg_pos_cvt.addDataSet(hi_theta_p_pos_cvt, 5);
dg_pos_cvt.addDataSet(hi_theta_phi_pos_cvt, 6);
dataGroups.add(dg_pos_cvt, 6);
//CVT Vertex
H2F hi_vxy_pos_cvt = new H2F("hi_vxy_pos_cvt","hi_vxy_pos_cvt",100,-.25,.25,100,-.25,.25);
hi_vxy_pos_cvt.setTitleX("Vx (cm)");
hi_vxy_pos_cvt.setTitleY("Vy (cm)");
H2F hi_vxy_neg_cvt = new H2F("hi_vxy_neg_cvt","hi_vxy_neg_cvt",100,-.25,.25,100,-.25,.25);
hi_vxy_neg_cvt.setTitleX("Vx (cm)");
hi_vxy_neg_cvt.setTitleY("Vy (cm)");
H2F hi_vz_vs_theta_pos_cvt = new H2F("hi_vz_vs_theta_pos_cvt","hi_vz_vs_theta_pos_cvt",100,
5.,120.,100,-15.,15);
hi_vz_vs_theta_pos_cvt.setTitleX("#theta (deg)");
hi_vz_vs_theta_pos_cvt.setTitleY("Vz (cm)");
```

```
H2F hi_vz_vs_theta_neg_cvt = new H2F("hi_vz_vs_theta_neg_cvt","hi_vz_vs_theta_neg_cvt",100, 5.,120.,100,-15.,15);

hi_vz_vs_theta_neg_cvt.setTitleX("#theta (deg)");

hi_vz_vs_theta_neg_cvt.setTitleY("Vz (cm)");

DataGroup cvtVertex = new DataGroup(2,2);

cvtVertex.addDataSet(hi_vz_vs_theta_pos_cvt, 0);

cvtVertex.addDataSet(hi_vxy_pos_cvt, 1);

cvtVertex.addDataSet(hi_vz_vs_theta_neg_cvt, 2);

cvtVertex.addDataSet(hi_vxy_neg_cvt, 3);

dataGroups.add(cvtVertex, 7);


// negative and positive tracks by sector

H1F hi_vz_neg_sec_1 = new H1F("hi_vz_neg_sec_1", "hi_vz_neg_sec_1", 100, -15.0, 15.0);

hi_vz_neg_sec_1.setTitleX("Vz (cm)");

hi_vz_neg_sec_1.setTitleY("Counts");

H1F hi_vz_neg_sec_1_cut = new H1F("hi_vz_neg_sec_1_cut", "hi_vz_neg_sec_1_cut", 100, -15.0, 15.0);

hi_vz_neg_sec_1_cut.setTitleX("Vz (cm)");

hi_vz_neg_sec_1_cut.setTitleY("Counts");

hi_vz_neg_sec_1_cut.setLineColor(2);

F1D f1_vz_neg_sec_1 = new F1D("f1_vz_neg_sec_1","[amp]*gaus(x,[mean],[sigma])", -5.0, 5.0);

f1_vz_neg_sec_1.setParameter(0, 0);

f1_vz_neg_sec_1.setParameter(1, 0);

f1_vz_neg_sec_1.setParameter(2, 1.0);

f1_vz_neg_sec_1.setLineWidth(2);

f1_vz_neg_sec_1.setLineColor(1);

f1_vz_neg_sec_1.setOptStat("1111");

H1F hi_vz_pos_sec_1 = new H1F("hi_vz_pos_sec_1", "hi_vz_pos_sec_1", 100, -15.0, 15.0);

hi_vz_pos_sec_1.setTitleX("Vz (cm)");
```

```
hi_vz_pos_sec_1.setTitleY("Counts");
H1F hi_vz_pos_sec_1_cut = new H1F("hi_vz_pos_sec_1_cut", "hi_vz_pos_sec_1_cut", 100, -15.0, 15.0);
hi_vz_pos_sec_1_cut.setTitleX("Vz (cm)");
hi_vz_pos_sec_1_cut.setTitleY("Counts");
hi_vz_pos_sec_1_cut.setLineColor(2);
F1D f1_vz_pos_sec_1 = new F1D("f1_vz_pos_sec_1","[amp]*gaus(x,[mean],[sigma])", -5.0, 5.0);
f1_vz_pos_sec_1.setParameter(0, 0);
f1_vz_pos_sec_1.setParameter(1, 0);
f1_vz_pos_sec_1.setParameter(2, 1.0);
f1_vz_pos_sec_1.setLineWidth(2);
f1_vz_pos_sec_1.setLineColor(1);
f1_vz_pos_sec_1.setOptStat("1111");
H1F hi_vz_neg_sec_2 = new H1F("hi_vz_neg_sec_2", "hi_vz_neg_sec_2", 100, -15.0, 15.0);
hi_vz_neg_sec_2.setTitleX("Vz (cm)");
hi_vz_neg_sec_2.setTitleY("Counts");
H1F hi_vz_neg_sec_2_cut = new H1F("hi_vz_neg_sec_2_cut", "hi_vz_neg_sec_2_cut", 100, -15.0, 15.0);
hi_vz_neg_sec_2_cut.setTitleX("Vz (cm)");
hi_vz_neg_sec_2_cut.setTitleY("Counts");
hi_vz_neg_sec_2_cut.setLineColor(2);
F1D f1_vz_neg_sec_2 = new F1D("f1_vz_neg_sec_2","[amp]*gaus(x,[mean],[sigma])", -5.0, 5.0);
f1_vz_neg_sec_2.setParameter(0, 0);
f1_vz_neg_sec_2.setParameter(1, 0);
f1_vz_neg_sec_2.setParameter(2, 1.0);
f1_vz_neg_sec_2.setLineWidth(2);
f1_vz_neg_sec_2.setLineColor(3);
f1_vz_neg_sec_2.setOptStat("1111");
H1F hi_vz_pos_sec_2 = new H1F("hi_vz_pos_sec_2", "hi_vz_pos_sec_2", 100, -15.0, 15.0);
```

```
hi_vz_pos_sec_2.setTitleX("Vz (cm)");

hi_vz_pos_sec_2.setTitleY("Counts");

H1F hi_vz_pos_sec_2_cut = new H1F("hi_vz_pos_sec_2_cut", "hi_vz_pos_sec_2_cut", 100, -15.0, 15.0);

hi_vz_pos_sec_2_cut.setTitleX("Vz (cm)");

hi_vz_pos_sec_2_cut.setTitleY("Counts");

hi_vz_pos_sec_2_cut.setLineColor(2);

F1D f1_vz_pos_sec_2 = new F1D("f1_vz_pos_sec_2","[amp]*gaus(x,[mean],[sigma])", -5.0, 5.0);

f1_vz_pos_sec_2.setParameter(0, 0);

f1_vz_pos_sec_2.setParameter(1, 0);

f1_vz_pos_sec_2.setParameter(2, 1.0);

f1_vz_pos_sec_2.setLineWidth(2);

f1_vz_pos_sec_2.setLineColor(3);

f1_vz_pos_sec_2.setOptStat("1111");

H1F hi_vz_neg_sec_3 = new H1F("hi_vz_neg_sec_3", "hi_vz_neg_sec_3", 100, -15.0, 15.0);

hi_vz_neg_sec_3.setTitleX("Vz (cm)");

hi_vz_neg_sec_3.setTitleY("Counts");

H1F hi_vz_neg_sec_3_cut = new H1F("hi_vz_neg_sec_3_cut", "hi_vz_neg_sec_3_cut", 100, -15.0, 15.0);

hi_vz_neg_sec_3_cut.setTitleX("Vz (cm)");

hi_vz_neg_sec_3_cut.setTitleY("Counts");

hi_vz_neg_sec_3_cut.setLineColor(2);

F1D f1_vz_neg_sec_3 = new F1D("f1_vz_neg_sec_3","[amp]*gaus(x,[mean],[sigma])", -5.0, 5.0);

f1_vz_neg_sec_3.setParameter(0, 0);

f1_vz_neg_sec_3.setParameter(1, 0);

f1_vz_neg_sec_3.setParameter(2, 1.0);

f1_vz_neg_sec_3.setLineWidth(2);

f1_vz_neg_sec_3.setLineColor(4);

f1_vz_neg_sec_3.setOptStat("1111");
```

```
H1F hi_vz_pos_sec_3 = new H1F("hi_vz_pos_sec_3", "hi_vz_pos_sec_3", 100, -15.0, 15.0);

hi_vz_pos_sec_3.setTitleX("Vz (cm)");

hi_vz_pos_sec_3.setTitleY("Counts");

H1F hi_vz_pos_sec_3_cut = new H1F("hi_vz_pos_sec_3_cut", "hi_vz_pos_sec_3_cut", 100, -15.0,
15.0);

hi_vz_pos_sec_3_cut.setTitleX("Vz (cm)");

hi_vz_pos_sec_3_cut.setTitleY("Counts");

hi_vz_pos_sec_3_cut.setLineColor(2);

F1D f1_vz_pos_sec_3 = new F1D("f1_vz_pos_sec_3","[amp]*gaus(x,[mean],[sigma])", -5.0, 5.0);

f1_vz_pos_sec_3.setParameter(0, 0);

f1_vz_pos_sec_3.setParameter(1, 0);

f1_vz_pos_sec_3.setParameter(2, 1.0);

f1_vz_pos_sec_3.setLineWidth(2);

f1_vz_pos_sec_3.setLineColor(4);

f1_vz_pos_sec_3.setOptStat("1111");

H1F hi_vz_neg_sec_4 = new H1F("hi_vz_neg_sec_4", "hi_vz_neg_sec_4", 100, -15.0, 15.0);

hi_vz_neg_sec_4.setTitleX("Vz (cm)");

hi_vz_neg_sec_4.setTitleY("Counts");

H1F hi_vz_neg_sec_4_cut = new H1F("hi_vz_neg_sec_4_cut", "hi_vz_neg_sec_4_cut", 100, -15.0,
15.0);

hi_vz_neg_sec_4_cut.setTitleX("Vz (cm)");

hi_vz_neg_sec_4_cut.setTitleY("Counts");

hi_vz_neg_sec_4_cut.setLineColor(2);

F1D f1_vz_neg_sec_4 = new F1D("f1_vz_neg_sec_4","[amp]*gaus(x,[mean],[sigma])", -5.0, 5.0);

f1_vz_neg_sec_4.setParameter(0, 0);

f1_vz_neg_sec_4.setParameter(1, 0);

f1_vz_neg_sec_4.setParameter(2, 1.0);

f1_vz_neg_sec_4.setLineWidth(2);

f1_vz_neg_sec_4.setLineColor(5);
```

```
f1_vz_neg_sec_4.setOptStat("1111");
H1F hi_vz_pos_sec_4 = new H1F("hi_vz_pos_sec_4", "hi_vz_pos_sec_4", 100, -15.0, 15.0);
hi_vz_pos_sec_4.setTitleX("Vz (cm)");
hi_vz_pos_sec_4.setTitleY("Counts");
H1F hi_vz_pos_sec_4_cut = new H1F("hi_vz_pos_sec_4_cut", "hi_vz_pos_sec_4_cut", 100, -15.0,
15.0);
hi_vz_pos_sec_4_cut.setTitleX("Vz (cm)");
hi_vz_pos_sec_4_cut.setTitleY("Counts");
hi_vz_pos_sec_4_cut.setLineColor(2);
F1D f1_vz_pos_sec_4 = new F1D("f1_vz_pos_sec_4","[amp]*gaus(x,[mean],[sigma])", -5.0, 5.0);
f1_vz_pos_sec_4.setParameter(0, 0);
f1_vz_pos_sec_4.setParameter(1, 0);
f1_vz_pos_sec_4.setParameter(2, 1.0);
f1_vz_pos_sec_4.setLineWidth(2);
f1_vz_pos_sec_4.setLineColor(5);
f1_vz_pos_sec_4.setOptStat("1111");
H1F hi_vz_neg_sec_5 = new H1F("hi_vz_neg_sec_5", "hi_vz_neg_sec_5", 100, -15.0, 15.0);
hi_vz_neg_sec_5.setTitleX("Vz (cm)");
hi_vz_neg_sec_5.setTitleY("Counts");
H1F hi_vz_neg_sec_5_cut = new H1F("hi_vz_neg_sec_5_cut", "hi_vz_neg_sec_5_cut", 100, -15.0,
15.0);
hi_vz_neg_sec_5_cut.setTitleX("Vz (cm)");
hi_vz_neg_sec_5_cut.setTitleY("Counts");
hi_vz_neg_sec_5_cut.setLineColor(2);
F1D f1_vz_neg_sec_5 = new F1D("f1_vz_neg_sec_5","[amp]*gaus(x,[mean],[sigma])", -5.0, 5.0);
f1_vz_neg_sec_5.setParameter(0, 0);
f1_vz_neg_sec_5.setParameter(1, 0);
f1_vz_neg_sec_5.setParameter(2, 1.0);
f1_vz_neg_sec_5.setLineWidth(2);
```

```
f1_vz_neg_sec_5.setLineColor(6);

f1_vz_neg_sec_5.setOptStat("1111");

H1F hi_vz_pos_sec_5 = new H1F("hi_vz_pos_sec_5", "hi_vz_pos_sec_5", 100, -15.0, 15.0);

hi_vz_pos_sec_5.setTitleX("Vz (cm)");

hi_vz_pos_sec_5.setTitleY("Counts");

H1F hi_vz_pos_sec_5_cut = new H1F("hi_vz_pos_sec_5_cut", "hi_vz_pos_sec_5_cut", 100, -15.0,
15.0);

hi_vz_pos_sec_5_cut.setTitleX("Vz (cm)");

hi_vz_pos_sec_5_cut.setTitleY("Counts");

hi_vz_pos_sec_5_cut.setLineColor(2);

F1D f1_vz_pos_sec_5 = new F1D("f1_vz_pos_sec_5","[amp]*gaus(x,[mean],[sigma])", -5.0, 5.0);

f1_vz_pos_sec_5.setParameter(0, 0);

f1_vz_pos_sec_5.setParameter(1, 0);

f1_vz_pos_sec_5.setParameter(2, 1.0);

f1_vz_pos_sec_5.setLineWidth(2);

f1_vz_pos_sec_5.setLineColor(6);

f1_vz_pos_sec_5.setOptStat("1111");

H1F hi_vz_neg_sec_6 = new H1F("hi_vz_neg_sec_6", "hi_vz_neg_sec_6", 100, -15.0, 15.0);

hi_vz_neg_sec_6.setTitleX("Vz (cm)");

hi_vz_neg_sec_6.setTitleY("Counts");

H1F hi_vz_neg_sec_6_cut = new H1F("hi_vz_neg_sec_6_cut", "hi_vz_neg_sec_6_cut", 100, -15.0,
15.0);

hi_vz_neg_sec_6_cut.setTitleX("Vz (cm)");

hi_vz_neg_sec_6_cut.setTitleY("Counts");

hi_vz_neg_sec_6_cut.setLineColor(2);

F1D f1_vz_neg_sec_6 = new F1D("f1_vz_neg_sec_6","[amp]*gaus(x,[mean],[sigma])", -5.0, 5.0);

f1_vz_neg_sec_6.setParameter(0, 0);

f1_vz_neg_sec_6.setParameter(1, 0);

f1_vz_neg_sec_6.setParameter(2, 1.0);
```

```
f1_vz_neg_sec_6.setLineWidth(2);
f1_vz_neg_sec_6.setLineColor(7);
f1_vz_neg_sec_6.setOptStat("1111");
H1F hi_vz_pos_sec_6 = new H1F("hi_vz_pos_sec_6", "hi_vz_pos_sec_6", 100, -15.0, 15.0);
hi_vz_pos_sec_6.setTitleX("Vz (cm)");
hi_vz_pos_sec_6.setTitleY("Counts");
H1F hi_vz_pos_sec_6_cut = new H1F("hi_vz_pos_sec_6_cut", "hi_vz_pos_sec_6_cut", 100, -15.0,
15.0);
hi_vz_pos_sec_6_cut.setTitleX("Vz (cm)");
hi_vz_pos_sec_6_cut.setTitleY("Counts");
hi_vz_pos_sec_6_cut.setLineColor(2);
F1D f1_vz_pos_sec_6 = new F1D("f1_vz_pos_sec_6","[amp]*gaus(x,[mean],[sigma])", -5.0, 5.0);
f1_vz_pos_sec_6.setParameter(0, 0);
f1_vz_pos_sec_6.setParameter(1, 0);
f1_vz_pos_sec_6.setParameter(2, 1.0);
f1_vz_pos_sec_6.setLineWidth(2);
f1_vz_pos_sec_6.setLineColor(7);
f1_vz_pos_sec_6.setOptStat("1111");

DataGroup dg_pos_sec = new DataGroup(3,2);
dg_pos_sec.addDataSet(hi_vz_pos_sec_1, 0);
dg_pos_sec.addDataSet(hi_vz_pos_sec_1_cut, 0);
dg_pos_sec.addDataSet(f1_vz_pos_sec_1, 0);
dg_pos_sec.addDataSet(hi_vz_pos_sec_2, 1);
dg_pos_sec.addDataSet(hi_vz_pos_sec_2_cut, 1);
dg_pos_sec.addDataSet(f1_vz_pos_sec_2, 1);
dg_pos_sec.addDataSet(hi_vz_pos_sec_3, 2);
dg_pos_sec.addDataSet(hi_vz_pos_sec_3_cut, 2);
dg_pos_sec.addDataSet(f1_vz_pos_sec_3, 2);
```

```
dg_pos_sec.addDataSet(hi_vz_pos_sec_4, 3);
dg_pos_sec.addDataSet(hi_vz_pos_sec_4_cut, 3);
dg_pos_sec.addDataSet(f1_vz_pos_sec_4, 3);
dg_pos_sec.addDataSet(hi_vz_pos_sec_5, 4);
dg_pos_sec.addDataSet(hi_vz_pos_sec_5_cut, 4);
dg_pos_sec.addDataSet(f1_vz_pos_sec_5, 4);
dg_pos_sec.addDataSet(hi_vz_pos_sec_6, 5);
dg_pos_sec.addDataSet(hi_vz_pos_sec_6_cut, 5);
dg_pos_sec.addDataSet(f1_vz_pos_sec_6, 5);
dataGroups.add(dg_pos_sec, 8);

DataGroup dg_neg_sec = new DataGroup(3,2);
dg_neg_sec.addDataSet(hi_vz_neg_sec_1, 0);
dg_neg_sec.addDataSet(hi_vz_neg_sec_1_cut, 0);
dg_neg_sec.addDataSet(f1_vz_neg_sec_1, 0);
dg_neg_sec.addDataSet(hi_vz_neg_sec_2, 1);
dg_neg_sec.addDataSet(hi_vz_neg_sec_2_cut, 1);
dg_neg_sec.addDataSet(f1_vz_neg_sec_2, 1);
dg_neg_sec.addDataSet(hi_vz_neg_sec_3, 2);
dg_neg_sec.addDataSet(hi_vz_neg_sec_3_cut, 2);
dg_neg_sec.addDataSet(f1_vz_neg_sec_3, 2);
dg_neg_sec.addDataSet(hi_vz_neg_sec_4, 3);
dg_neg_sec.addDataSet(hi_vz_neg_sec_4_cut, 3);
dg_neg_sec.addDataSet(f1_vz_neg_sec_4, 3);
dg_neg_sec.addDataSet(hi_vz_neg_sec_5, 4);
dg_neg_sec.addDataSet(hi_vz_neg_sec_5_cut, 4);
dg_neg_sec.addDataSet(f1_vz_neg_sec_5, 4);
dg_neg_sec.addDataSet(hi_vz_neg_sec_6, 5);
dg_neg_sec.addDataSet(hi_vz_neg_sec_6_cut, 5);
```

```
dg_neg_sec.addDataSet(f1_vz_neg_sec_6, 5);

dataGroups.add(dg_neg_sec, 9);



H2F vertex_CVT_TBT_pos = new H2F("vertex_CVT_TBT_pos","vertex_CVT_TBT_pos",100,
0.,40.,100,-15.,15);

vertex_CVT_TBT_pos.setTitleX("Vz CVT (cm)");

vertex_CVT_TBT_pos.setTitleY("Vz TBT (cm)");

H1F vertex_CVT_TBT_diff_pos = new H1F("vertex_CVT_TBT_diff_pos", "vertex_CVT_TBT_diff_pos",
100, -15., 15.);

vertex_CVT_TBT_diff_pos.setTitleX("Vz (cm)");

vertex_CVT_TBT_diff_pos.setTitleY("Counts");

H2F vertex_CVT_TBT_neg = new H2F("vertex_CVT_TBT_neg","vertex_CVT_TBT_neg",100,
0.,40.,100,-15.,15);

vertex_CVT_TBT_neg.setTitleX("Vz CVT (cm)");

vertex_CVT_TBT_neg.setTitleY("Vz TBT (cm)");

H1F vertex_CVT_TBT_diff_neg = new H1F("vertex_CVT_TBT_diff_neg", "vertex_CVT_TBT_diff_neg",
100, -15., 15.);

vertex_CVT_TBT_diff_neg.setTitleX("Vz (cm)");

vertex_CVT_TBT_diff_neg.setTitleY("Counts");

DataGroup vertex_comparison = new DataGroup(2,2);

vertex_comparison.addDataSet(vertex_CVT_TBT_pos, 0);

vertex_comparison.addDataSet(vertex_CVT_TBT_diff_pos, 1);

vertex_comparison.addDataSet(vertex_CVT_TBT_neg, 2);

vertex_comparison.addDataSet(vertex_CVT_TBT_diff_neg, 3);

dataGroups.add(vertex_comparison, 10);


DataGroup dg_neg_sec_main = new DataGroup(1,1);

dg_neg_sec_main.addDataSet(f1_vz_neg_sec_1, 0);

dg_neg_sec_main.addDataSet(f1_vz_neg_sec_2, 0);
```

```
dg_neg_sec_main.addDataSet(f1_vz_neg_sec_3, 0);
dg_neg_sec_main.addDataSet(f1_vz_neg_sec_4, 0);
dg_neg_sec_main.addDataSet(f1_vz_neg_sec_5, 0);
dg_neg_sec_main.addDataSet(f1_vz_neg_sec_6, 0);
dataGroups.add(dg_neg_sec_main, 11);

DataGroup dg_pos_sec_main = new DataGroup(1,1);
dg_pos_sec_main.addDataSet(f1_vz_pos_sec_1, 0);
dg_pos_sec_main.addDataSet(f1_vz_pos_sec_2, 0);
dg_pos_sec_main.addDataSet(f1_vz_pos_sec_3, 0);
dg_pos_sec_main.addDataSet(f1_vz_pos_sec_4, 0);
dg_pos_sec_main.addDataSet(f1_vz_pos_sec_5, 0);
dg_pos_sec_main.addDataSet(f1_vz_pos_sec_6, 0);
dataGroups.add(dg_pos_sec_main, 12);

H2F p_vs_beta_pos = new H2F("p_vs_beta_pos", "p_vs_beta_pos", 100, 0., 4., 100, 0., 1.2);
p_vs_beta_pos.setTitleX("P");
p_vs_beta_pos.setTitleY("beta");
p_vs_beta_pos.setTitle("Forward Detector Positive Tracks");
H2F p_vs_beta_neg = new H2F("p_vs_beta_neg", "p_vs_beta_neg", 100, 0., 4., 100, 0., 1.2);
p_vs_beta_neg.setTitleX("P");
p_vs_beta_neg.setTitleY("beta");
p_vs_beta_neg.setTitle("Forward Detector Negative Tracks");
H2F p_vs_beta_cvt_pos = new H2F("p_vs_beta_cvt_pos", "p_vs_beta_cvt_pos", 100, 0., 4., 100, 0.,
1.2);
p_vs_beta_cvt_pos.setTitleX("P");
p_vs_beta_cvt_pos.setTitleY("beta");
p_vs_beta_cvt_pos.setTitle("CVT Positive Tracks");
```

```
H2F p_vs_beta_cvt_neg = new H2F("p_vs_beta_cvt_neg", "p_vs_beta_cvt_neg", 100, 0., 4., 100, 0., 1.2);

p_vs_beta_cvt_neg.setTitleX("P");

p_vs_beta_cvt_neg.setTitleY("beta");

p_vs_beta_cvt_neg.setTitle("CVT Negative Tracks");

H1F neg_particle_count = new H1F("neg_particle_count", "neg_particle_count", 100, 0.5, 1.5);

neg_particle_count.setTitleX("Beta");

neg_particle_count.setTitleY("Counts");

neg_particle_count.setTitle("Forward Detector Negative Tracks Histogram");

H1F neg_particle_cvt_count = new H1F("neg_particle_cvt_count", "neg_particle_cvt_count", 100, 0.5, 1.5);

neg_particle_cvt_count.setTitleX("Beta");

neg_particle_cvt_count.setTitleY("Counts");

neg_particle_cvt_count.setTitle("CVT Negative Tracks Histogram");

H1F pos_particle_count = new H1F("pos_particle_count", "pos_particle_count", 100, 0.5, 1.5);

pos_particle_count.setTitleX("Beta");

pos_particle_count.setTitleY("Counts");

pos_particle_count.setTitle("Forward Detector Positive Tracks Histogram");

H1F pos_particle_cvt_count = new H1F("pos_particle_cvt_count", "pos_particle_cvt_count", 100, 0.5, 1.5);

pos_particle_cvt_count.setTitleX("Beta");

pos_particle_cvt_count.setTitleY("Counts");

pos_particle_cvt_count.setTitle("CVT Positive Tracks Histogram");

DataGroup beta = new DataGroup(2,4);

beta.addDataSet(p_vs_beta_pos, 0);

beta.addDataSet(p_vs_beta_neg, 1);

beta.addDataSet(p_vs_beta_cvt_neg, 2);

beta.addDataSet(p_vs_beta_cvt_pos, 3);

beta.addDataSet(neg_particle_count,4);

beta.addDataSet(neg_particle_cvt_count,5);
```

```
beta.addDataSet(pos_particle_count,6);
beta.addDataSet(pos_particle_cvt_count,7);
dataGroups.add(beta, 13);



H2F p_vs_Beta_neg = new H2F("p_vs_Beta_neg", "p_vs_Beta_neg", 100, 0., 4., 100, 0., 1.2);
p_vs_Beta_neg.setTitleX("P");
p_vs_Beta_neg.setTitleY("beta");
p_vs_Beta_neg.setTitle("Forward Detector Negative Tracks");
H2F p_vs_Beta_pos = new H2F("p_vs_Beta_pos", "p_vs_Beta_pos", 100, 0., 4., 100, 0., 1.2);
p_vs_Beta_pos.setTitleX("P");
p_vs_Beta_pos.setTitleY("beta");
p_vs_Beta_pos.setTitle("Forward Detector Negative Tracks");
H1F v_time_neg = new H1F("v_time_neg", "v_time_neg", 200, -3.0, 3.0);
v_time_neg.setTitleX("Delta t (ns)");
v_time_neg.setTitleY("Counts");
v_time_neg.setTitle("Forward Detector Negative Vertex Time Histogram");
H1F v_time_corr_neg = new H1F("v_time_corr_neg", "v_time_corr_neg", 200, -3.0, 3.0);
v_time_corr_neg.setLineColor(2);
v_time_corr_neg.setTitleX("Delta t (ns)");
v_time_corr_neg.setTitleY("Counts");
v_time_corr_neg.setTitle("Forward Detector Negative Vertex Time Histogram");



DataGroup BetaCalc = new DataGroup(2,2);
BetaCalc.addDataSet(p_vs_Beta_neg, 0);
BetaCalc.addDataSet(p_vs_Beta_pos, 1);
BetaCalc.addDataSet(v_time_neg, 2);
```

```
    BetaCalc.addDataSet(v_time_corr_neg, 2);
    dataGroups.add(BetaCalc, 14);
}

//plotHistos produces plots from those specfied in createHistos().

private void plotHistos() {
    canvasTabbed.getCanvas("Negative Tracks").divide(4,2);
    canvasTabbed.getCanvas("Negative Tracks").setGridX(false);
    canvasTabbed.getCanvas("Negative Tracks").setGridY(false);
    canvasTabbed.getCanvas("Positive Tracks").divide(4,2);
    canvasTabbed.getCanvas("Positive Tracks").setGridX(false);
    canvasTabbed.getCanvas("Positive Tracks").setGridY(false);
    canvasTabbed.getCanvas("Monte Carlo").divide(4, 2);
    canvasTabbed.getCanvas("Monte Carlo").setGridX(false);
    canvasTabbed.getCanvas("Monte Carlo").setGridY(false);
    canvasTabbed.getCanvas("Monte Carlo p negative").divide(3, 2);
    canvasTabbed.getCanvas("Monte Carlo p negative").setGridX(false);
    canvasTabbed.getCanvas("Monte Carlo p negative").setGridY(false);
    canvasTabbed.getCanvas("Vertex").divide(3, 2);
    canvasTabbed.getCanvas("Vertex").setGridX(false);
    canvasTabbed.getCanvas("Vertex").setGridY(false);
    canvasTabbed.getCanvas("CVT Negative Tracks").divide(3,2);
    canvasTabbed.getCanvas("CVT Negative Tracks").setGridX(false);
    canvasTabbed.getCanvas("CVT Negative Tracks").setGridY(false);

    canvasTabbed.getCanvas("CVT Positive Tracks").divide(3,2);
    canvasTabbed.getCanvas("CVT Positive Tracks").setGridX(false);
    canvasTabbed.getCanvas("CVT Positive Tracks").setGridY(false);
```

```
canvasTabbed.getCanvas("CVT Vertex").divide(3, 2);
canvasTabbed.getCanvas("CVT Vertex").setGridX(false);
canvasTabbed.getCanvas("CVT Vertex").setGridY(false);


canvasTabbed.getCanvas("Positive Tracks by Sector").divide(3,2);
canvasTabbed.getCanvas("Positive Tracks by Sector").setGridX(false);
canvasTabbed.getCanvas("Positive Tracks by Sector").setGridY(false);

canvasTabbed.getCanvas("Negative Tracks by Sector").divide(3,2);
canvasTabbed.getCanvas("Negative Tracks by Sector").setGridX(false);
canvasTabbed.getCanvas("Negative Tracks by Sector").setGridY(false);

canvasTabbed.getCanvas("Negative Tracks by Sector Main").divide(1,1);
canvasTabbed.getCanvas("Negative Tracks by Sector Main").setGridX(false);
canvasTabbed.getCanvas("Negative Tracks by Sector Main").setGridY(false);

canvasTabbed.getCanvas("Vertex Comparison").divide(2, 2);
canvasTabbed.getCanvas("Vertex Comparison").setGridX(false);
canvasTabbed.getCanvas("Vertex Comparison").setGridY(false);

canvasTabbed.getCanvas("beta").divide(2, 4);
canvasTabbed.getCanvas("beta").setGridX(false);
canvasTabbed.getCanvas("beta").setGridY(false);

canvasTabbed.getCanvas("Calculated Beta").divide(2, 2);
canvasTabbed.getCanvas("Calculated Beta").setGridX(false);
canvasTabbed.getCanvas("Calculated Beta").setGridY(false);
```

```
        canvasTabbed.getCanvas("Negative Tracks").cd(0);
        canvasTabbed.getCanvas("Negative Tracks").draw(dataGroups.getItem(1).getH1F("hi_p_neg"));
        canvasTabbed.getCanvas("Negative Tracks").cd(1);
        canvasTabbed.getCanvas("Negative Tracks").draw(dataGroups.getItem(1).getH1F("hi_theta_neg"));
        canvasTabbed.getCanvas("Negative Tracks").cd(2);
        canvasTabbed.getCanvas("Negative Tracks").draw(dataGroups.getItem(1).getH1F("hi_phi_neg"));
        canvasTabbed.getCanvas("Negative Tracks").cd(3);
        canvasTabbed.getCanvas("Negative Tracks").draw(dataGroups.getItem(1).getH1F("hi_chi2_neg"));
        canvasTabbed.getCanvas("Negative Tracks").cd(4);
        canvasTabbed.getCanvas("Negative Tracks").draw(dataGroups.getItem(1).getH1F("hi_vz_neg"));
        canvasTabbed.getCanvas("Negative
Tracks").draw(dataGroups.getItem(1).getH1F("hi_vz_neg_cut"),"same");
        canvasTabbed.getCanvas("Negative
Tracks").draw(dataGroups.getItem(1).getF1D("f1_vz_neg"),"same");
        canvasTabbed.getCanvas("Negative Tracks").cd(5);
        canvasTabbed.getCanvas("Negative
Tracks").draw(dataGroups.getItem(1).getH2F("hi_theta_p_neg"));
        canvasTabbed.getCanvas("Negative Tracks").cd(6);
        canvasTabbed.getCanvas("Negative
Tracks").draw(dataGroups.getItem(1).getH2F("hi_theta_phi_neg"));
        canvasTabbed.getCanvas("Negative Tracks").cd(7);
        canvasTabbed.getCanvas("Negative
Tracks").draw(dataGroups.getItem(1).getH2F("hi_chi2_vz_neg"));
        canvasTabbed.getCanvas("Positive Tracks").cd(0);
        canvasTabbed.getCanvas("Positive Tracks").draw(dataGroups.getItem(2).getH1F("hi_p_pos"));
        canvasTabbed.getCanvas("Positive Tracks").cd(1);
        canvasTabbed.getCanvas("Positive Tracks").draw(dataGroups.getItem(2).getH1F("hi_theta_pos"));
        canvasTabbed.getCanvas("Positive Tracks").cd(2);
        canvasTabbed.getCanvas("Positive Tracks").draw(dataGroups.getItem(2).getH1F("hi_phi_pos"));
        canvasTabbed.getCanvas("Positive Tracks").cd(3);
```

```
        canvasTabbed.getCanvas("Positive Tracks").draw(dataGroups.getItem(2).getH1F("hi_chi2_pos"));

        canvasTabbed.getCanvas("Positive Tracks").cd(4);

        canvasTabbed.getCanvas("Positive Tracks").draw(dataGroups.getItem(2).getH1F("hi_vz_pos"));

        canvasTabbed.getCanvas("Positive
Tracks").draw(dataGroups.getItem(2).getH1F("hi_vz_pos_cut"),"same");

        canvasTabbed.getCanvas("Positive
Tracks").draw(dataGroups.getItem(2).getF1D("f1_vz_pos"),"same");

        canvasTabbed.getCanvas("Positive Tracks").cd(5);

        canvasTabbed.getCanvas("Positive
Tracks").draw(dataGroups.getItem(2).getH2F("hi_theta_p_pos"));

        canvasTabbed.getCanvas("Positive Tracks").cd(6);

        canvasTabbed.getCanvas("Positive
Tracks").draw(dataGroups.getItem(2).getH2F("hi_theta_phi_pos"));

        canvasTabbed.getCanvas("Positive Tracks").cd(7);

        canvasTabbed.getCanvas("Positive
Tracks").draw(dataGroups.getItem(2).getH2F("hi_chi2_vz_pos"));

        canvasTabbed.getCanvas("Monte Carlo").cd(0);

        canvasTabbed.getCanvas("Monte Carlo").draw(dataGroups.getItem(3).getH1F("hi_dp_pos"));

        canvasTabbed.getCanvas("Monte
Carlo").draw(dataGroups.getItem(3).getF1D("f1_dp_pos"),"same");

        canvasTabbed.getCanvas("Monte Carlo").cd(1);

        canvasTabbed.getCanvas("Monte Carlo").draw(dataGroups.getItem(3).getH1F("hi_dtheta_pos"));

        canvasTabbed.getCanvas("Monte
Carlo").draw(dataGroups.getItem(3).getF1D("f1_dtheta_pos"),"same");

        canvasTabbed.getCanvas("Monte Carlo").cd(2);

        canvasTabbed.getCanvas("Monte Carlo").draw(dataGroups.getItem(3).getH1F("hi_dphi_pos"));

        canvasTabbed.getCanvas("Monte
Carlo").draw(dataGroups.getItem(3).getF1D("f1_dphi_pos"),"same");

        canvasTabbed.getCanvas("Monte Carlo").cd(3);

        canvasTabbed.getCanvas("Monte Carlo").draw(dataGroups.getItem(3).getH1F("hi_dvz_pos"));

        canvasTabbed.getCanvas("Monte
Carlo").draw(dataGroups.getItem(3).getF1D("f1_dvz_pos"),"same");
```

```
        canvasTabbed.getCanvas("Monte Carlo").cd(4);

        canvasTabbed.getCanvas("Monte Carlo").draw(dataGroups.getItem(3).getH1F("hi_dp_neg"));

        canvasTabbed.getCanvas("Monte
Carlo").draw(dataGroups.getItem(3).getF1D("f1_dp_neg"),"same");

        canvasTabbed.getCanvas("Monte Carlo").cd(5);

        canvasTabbed.getCanvas("Monte Carlo").draw(dataGroups.getItem(3).getH1F("hi_dtheta_neg"));

        canvasTabbed.getCanvas("Monte
Carlo").draw(dataGroups.getItem(3).getF1D("f1_dtheta_neg"),"same");

        canvasTabbed.getCanvas("Monte Carlo").cd(6);

        canvasTabbed.getCanvas("Monte Carlo").draw(dataGroups.getItem(3).getH1F("hi_dphi_neg"));

        canvasTabbed.getCanvas("Monte
Carlo").draw(dataGroups.getItem(3).getF1D("f1_dphi_neg"),"same");

        canvasTabbed.getCanvas("Monte Carlo").cd(7);

        canvasTabbed.getCanvas("Monte Carlo").draw(dataGroups.getItem(3).getH1F("hi_dvz_neg"));

        canvasTabbed.getCanvas("Monte
Carlo").draw(dataGroups.getItem(3).getF1D("f1_dvz_neg"),"same");

        canvasTabbed.getCanvas("Monte Carlo p negative").cd(0);

        canvasTabbed.getCanvas("Monte Carlo p
negative").draw(dataGroups.getItem(3).getH2F("hi_dp_p_neg"));

        canvasTabbed.getCanvas("Monte Carlo p negative").cd(1);

        canvasTabbed.getCanvas("Monte Carlo p
negative").draw(dataGroups.getItem(3).getH2F("hi_dp_theta_neg"));

        canvasTabbed.getCanvas("Monte Carlo p negative").cd(2);

        canvasTabbed.getCanvas("Monte Carlo p
negative").draw(dataGroups.getItem(3).getH2F("hi_dp_phi_neg"));

        canvasTabbed.getCanvas("Monte Carlo p negative").cd(3);

        canvasTabbed.getCanvas("Monte Carlo p negative").getPad(3).getAxisY().setRange(-0.05, 0.05);

        if(dataGroups.getItem(3).getGraph("gr_dp_p_neg").getDataSize(0)>1)
canvasTabbed.getCanvas("Monte Carlo p
negative").draw(dataGroups.getItem(3).getGraph("gr_dp_p_neg"));

        canvasTabbed.getCanvas("Monte Carlo p negative").cd(4);

        canvasTabbed.getCanvas("Monte Carlo p negative").getPad(4).getAxisY().setRange(-0.05, 0.05);
```

```
    if(dataGroups.getItem(3).getGraph("gr_dp_theta_neg").getDataSize(0)>1)
canvasTabbed.getCanvas("Monte Carlo p
negative").draw(dataGroups.getItem(3).getGraph("gr_dp_theta_neg"));

    canvasTabbed.getCanvas("Monte Carlo p negative").cd(5);

    canvasTabbed.getCanvas("Monte Carlo p negative").getPad(5).getAxisY().setRange(-0.05, 0.05);

    if(dataGroups.getItem(3).getGraph("gr_dp_phi_neg").getDataSize(0)>1)
canvasTabbed.getCanvas("Monte Carlo p
negative").draw(dataGroups.getItem(3).getGraph("gr_dp_phi_neg"));

    canvasTabbed.getCanvas("Vertex").cd(0);

    canvasTabbed.getCanvas("Vertex").draw(dataGroups.getItem(2).getH1F("hi_vz_pos"));

    canvasTabbed.getCanvas("Vertex").draw(dataGroups.getItem(2).getH1F("hi_vz_pos_cut"),"same");

    canvasTabbed.getCanvas("Vertex").draw(dataGroups.getItem(2).getF1D("f1_vz_pos"),"same");

    canvasTabbed.getCanvas("Vertex").cd(1);

    canvasTabbed.getCanvas("Vertex").draw(dataGroups.getItem(4).getH2F("hi_vz_vs_theta_pos"));

    canvasTabbed.getCanvas("Vertex").cd(2);

    canvasTabbed.getCanvas("Vertex").getPad(2).getAxisZ().setLog(true);

    canvasTabbed.getCanvas("Vertex").draw(dataGroups.getItem(4).getH2F("hi_vxy_pos"));

    canvasTabbed.getCanvas("Vertex").cd(3);

    canvasTabbed.getCanvas("Vertex").draw(dataGroups.getItem(1).getH1F("hi_vz_neg"));

    canvasTabbed.getCanvas("Vertex").draw(dataGroups.getItem(1).getH1F("hi_vz_neg_cut"),"same");

    canvasTabbed.getCanvas("Vertex").draw(dataGroups.getItem(1).getF1D("f1_vz_neg"),"same");

    canvasTabbed.getCanvas("Vertex").cd(4);

    canvasTabbed.getCanvas("Vertex").draw(dataGroups.getItem(4).getH2F("hi_vz_vs_theta_neg"));

    canvasTabbed.getCanvas("Vertex").cd(5);

    canvasTabbed.getCanvas("Vertex").getPad(5).getAxisZ().setLog(true);

    canvasTabbed.getCanvas("Vertex").draw(dataGroups.getItem(4).getH2F("hi_vxy_neg"));


    canvasTabbed.getCanvas("CVT Negative Tracks").cd(0);

    canvasTabbed.getCanvas("CVT Negative
Tracks").draw(dataGroups.getItem(5).getH1F("hi_p_neg_cvt"));
```

```
        canvasTabbed.getCanvas("CVT Negative Tracks").cd(1);

        canvasTabbed.getCanvas("CVT Negative
Tracks").draw(dataGroups.getItem(5).getH1F("hi_theta_neg_cvt"));

        canvasTabbed.getCanvas("CVT Negative Tracks").cd(2);

        canvasTabbed.getCanvas("CVT Negative
Tracks").draw(dataGroups.getItem(5).getH1F("hi_phi_neg_cvt"));

        canvasTabbed.getCanvas("CVT Negative Tracks").cd(3);

        canvasTabbed.getCanvas("CVT Negative
Tracks").draw(dataGroups.getItem(5).getH1F("hi_chi2_neg_cvt"));

        canvasTabbed.getCanvas("CVT Negative Tracks").cd(3);

        canvasTabbed.getCanvas("CVT Negative
Tracks").draw(dataGroups.getItem(5).getH1F("hi_vz_neg_cvt"));

        canvasTabbed.getCanvas("CVT Negative
Tracks").draw(dataGroups.getItem(5).getF1D("f1_vz_neg_cvt"),"same");

        canvasTabbed.getCanvas("CVT Negative Tracks").cd(4);

        canvasTabbed.getCanvas("CVT Negative
Tracks").draw(dataGroups.getItem(5).getH2F("hi_theta_p_neg_cvt"));

        canvasTabbed.getCanvas("CVT Negative Tracks").cd(5);

        canvasTabbed.getCanvas("CVT Negative
Tracks").draw(dataGroups.getItem(5).getH2F("hi_theta_phi_neg_cvt"));

        canvasTabbed.getCanvas("CVT Negative Tracks").cd(6);

        canvasTabbed.getCanvas("CVT Negative
Tracks").draw(dataGroups.getItem(5).getH2F("hi_chi2_vz_neg_cvt"));

        canvasTabbed.getCanvas("CVT Positive Tracks").cd(0);

        canvasTabbed.getCanvas("CVT Positive
Tracks").draw(dataGroups.getItem(6).getH1F("hi_p_pos_cvt"));

        canvasTabbed.getCanvas("CVT Positive Tracks").cd(1);

        canvasTabbed.getCanvas("CVT Positive
Tracks").draw(dataGroups.getItem(6).getH1F("hi_theta_pos_cvt"));

        canvasTabbed.getCanvas("CVT Positive Tracks").cd(2);

        canvasTabbed.getCanvas("CVT Positive
Tracks").draw(dataGroups.getItem(6).getH1F("hi_phi_pos_cvt"));

        canvasTabbed.getCanvas("CVT Positive Tracks").cd(3);
```

```
        canvasTabbed.getCanvas("CVT Positive
Tracks").draw(dataGroups.getItem(6).getH1F("hi_chi2_pos_cvt"));

        canvasTabbed.getCanvas("CVT Positive Tracks").cd(4);

        canvasTabbed.getCanvas("CVT Positive
Tracks").draw(dataGroups.getItem(6).getH1F("hi_vz_pos_cvt"));

        canvasTabbed.getCanvas("CVT Positive
Tracks").draw(dataGroups.getItem(6).getH1F("hi_vz_pos_cvt_cut"),"same");

        canvasTabbed.getCanvas("CVT Positive
Tracks").draw(dataGroups.getItem(6).getF1D("f1_vz_pos_cvt"),"same");

        canvasTabbed.getCanvas("CVT Positive Tracks").cd(5);

        canvasTabbed.getCanvas("CVT Positive
Tracks").draw(dataGroups.getItem(6).getH2F("hi_theta_p_pos_cvt"));

        canvasTabbed.getCanvas("CVT Positive Tracks").cd(6);

        canvasTabbed.getCanvas("CVT Positive
Tracks").draw(dataGroups.getItem(6).getH2F("hi_theta_phi_pos_cvt"));


        //CVT Vertex

        canvasTabbed.getCanvas("CVT Vertex").cd(0);

        canvasTabbed.getCanvas("CVT Vertex").draw(dataGroups.getItem(6).getH1F("hi_vz_pos_cvt"));

        canvasTabbed.getCanvas("CVT
Vertex").draw(dataGroups.getItem(6).getH1F("hi_vz_pos_cvt_cut"),"same");

        canvasTabbed.getCanvas("CVT
Vertex").draw(dataGroups.getItem(6).getF1D("f1_vz_pos_cvt"),"same");

        canvasTabbed.getCanvas("CVT Vertex").cd(1);

        canvasTabbed.getCanvas("CVT
Vertex").draw(dataGroups.getItem(7).getH2F("hi_vz_vs_theta_pos_cvt"));

        canvasTabbed.getCanvas("CVT Vertex").cd(2);

        canvasTabbed.getCanvas("CVT Vertex").getPad(2).getAxisZ().setLog(true);

        canvasTabbed.getCanvas("CVT Vertex").draw(dataGroups.getItem(7).getH2F("hi_vxy_pos_cvt"));

        canvasTabbed.getCanvas("CVT Vertex").cd(3);

        canvasTabbed.getCanvas("CVT Vertex").draw(dataGroups.getItem(5).getH1F("hi_vz_neg_cvt"));
```

```
        canvasTabbed.getCanvas("CVT
Vertex").draw(dataGroups.getItem(5).getF1D("f1_vz_neg_cvt"),"same");

        canvasTabbed.getCanvas("CVT Vertex").cd(4);

        canvasTabbed.getCanvas("CVT
Vertex").draw(dataGroups.getItem(7).getH2F("hi_vz_vs_theta_neg_cvt"));

        canvasTabbed.getCanvas("CVT Vertex").cd(5);

        canvasTabbed.getCanvas("CVT Vertex").getPad(5).getAxisZ().setLog(true);

        canvasTabbed.getCanvas("CVT Vertex").draw(dataGroups.getItem(7).getH2F("hi_vxy_neg_cvt"));

        canvasTabbed.getCanvas("Positive Tracks by Sector").cd(0);

        canvasTabbed.getCanvas("Positive Tracks by
Sector").draw(dataGroups.getItem(8).getH1F("hi_vz_pos_sec_1"));

        canvasTabbed.getCanvas("Positive Tracks by
Sector").draw(dataGroups.getItem(8).getH1F("hi_vz_pos_sec_1_cut"),"same");

        canvasTabbed.getCanvas("Positive Tracks by
Sector").draw(dataGroups.getItem(8).getF1D("f1_vz_pos_sec_1"),"same");

        canvasTabbed.getCanvas("Positive Tracks by Sector").cd(1);

        canvasTabbed.getCanvas("Positive Tracks by
Sector").draw(dataGroups.getItem(8).getH1F("hi_vz_pos_sec_2"));

        canvasTabbed.getCanvas("Positive Tracks by
Sector").draw(dataGroups.getItem(8).getH1F("hi_vz_pos_sec_2_cut"),"same");

        canvasTabbed.getCanvas("Positive Tracks by
Sector").draw(dataGroups.getItem(8).getF1D("f1_vz_pos_sec_2"),"same");

        canvasTabbed.getCanvas("Positive Tracks by Sector").cd(2);

        canvasTabbed.getCanvas("Positive Tracks by
Sector").draw(dataGroups.getItem(8).getH1F("hi_vz_pos_sec_3"));

        canvasTabbed.getCanvas("Positive Tracks by
Sector").draw(dataGroups.getItem(8).getH1F("hi_vz_pos_sec_3_cut"),"same");

        canvasTabbed.getCanvas("Positive Tracks by
Sector").draw(dataGroups.getItem(8).getF1D("f1_vz_pos_sec_3"),"same");

        canvasTabbed.getCanvas("Positive Tracks by Sector").cd(3);

        canvasTabbed.getCanvas("Positive Tracks by
Sector").draw(dataGroups.getItem(8).getH1F("hi_vz_pos_sec_4"));
```

```
        canvasTabbed.getCanvas("Positive Tracks by
Sector").draw(dataGroups.getItem(8).getH1F("hi_vz_pos_sec_4_cut"),"same");

        canvasTabbed.getCanvas("Positive Tracks by
Sector").draw(dataGroups.getItem(8).getF1D("f1_vz_pos_sec_4"),"same");

        canvasTabbed.getCanvas("Positive Tracks by Sector").cd(4);

        canvasTabbed.getCanvas("Positive Tracks by
Sector").draw(dataGroups.getItem(8).getH1F("hi_vz_pos_sec_5"));

        canvasTabbed.getCanvas("Positive Tracks by
Sector").draw(dataGroups.getItem(8).getH1F("hi_vz_pos_sec_5_cut"),"same");

        canvasTabbed.getCanvas("Positive Tracks by
Sector").draw(dataGroups.getItem(8).getF1D("f1_vz_pos_sec_5"),"same");

        canvasTabbed.getCanvas("Positive Tracks by Sector").cd(5);

        canvasTabbed.getCanvas("Positive Tracks by
Sector").draw(dataGroups.getItem(8).getH1F("hi_vz_pos_sec_6"));

        canvasTabbed.getCanvas("Positive Tracks by
Sector").draw(dataGroups.getItem(8).getH1F("hi_vz_pos_sec_6_cut"),"same");

        canvasTabbed.getCanvas("Positive Tracks by
Sector").draw(dataGroups.getItem(8).getF1D("f1_vz_pos_sec_6"),"same");


        canvasTabbed.getCanvas("Negative Tracks by Sector").cd(0);

        canvasTabbed.getCanvas("Negative Tracks by
Sector").draw(dataGroups.getItem(9).getH1F("hi_vz_neg_sec_1"));

        canvasTabbed.getCanvas("Negative Tracks by
Sector").draw(dataGroups.getItem(9).getH1F("hi_vz_neg_sec_1_cut"),"same");

        canvasTabbed.getCanvas("Negative Tracks by
Sector").draw(dataGroups.getItem(9).getF1D("f1_vz_neg_sec_1"),"same");

        canvasTabbed.getCanvas("Negative Tracks by Sector").cd(1);

        canvasTabbed.getCanvas("Negative Tracks by
Sector").draw(dataGroups.getItem(9).getH1F("hi_vz_neg_sec_2"));

        canvasTabbed.getCanvas("Negative Tracks by
Sector").draw(dataGroups.getItem(9).getH1F("hi_vz_neg_sec_2_cut"),"same");

        canvasTabbed.getCanvas("Negative Tracks by
Sector").draw(dataGroups.getItem(9).getF1D("f1_vz_neg_sec_2"),"same");

        canvasTabbed.getCanvas("Negative Tracks by Sector").cd(2);
```

```
        canvasTabbed.getCanvas("Negative Tracks by
Sector").draw(dataGroups.getItem(9).getH1F("hi_vz_neg_sec_3"));

        canvasTabbed.getCanvas("Negative Tracks by
Sector").draw(dataGroups.getItem(9).getH1F("hi_vz_neg_sec_3_cut"),"same");

        canvasTabbed.getCanvas("Negative Tracks by
Sector").draw(dataGroups.getItem(9).getF1D("f1_vz_neg_sec_3"),"same");

        canvasTabbed.getCanvas("Negative Tracks by Sector").cd(3);

        canvasTabbed.getCanvas("Negative Tracks by
Sector").draw(dataGroups.getItem(9).getH1F("hi_vz_neg_sec_4"));

        canvasTabbed.getCanvas("Negative Tracks by
Sector").draw(dataGroups.getItem(9).getH1F("hi_vz_neg_sec_4_cut"),"same");

        canvasTabbed.getCanvas("Negative Tracks by
Sector").draw(dataGroups.getItem(9).getF1D("f1_vz_neg_sec_4"),"same");

        canvasTabbed.getCanvas("Negative Tracks by Sector").cd(4);

        canvasTabbed.getCanvas("Negative Tracks by
Sector").draw(dataGroups.getItem(9).getH1F("hi_vz_neg_sec_5"));

        canvasTabbed.getCanvas("Negative Tracks by
Sector").draw(dataGroups.getItem(9).getH1F("hi_vz_neg_sec_5_cut"),"same");

        canvasTabbed.getCanvas("Negative Tracks by
Sector").draw(dataGroups.getItem(9).getF1D("f1_vz_neg_sec_5"),"same");

        canvasTabbed.getCanvas("Negative Tracks by Sector").cd(5);

        canvasTabbed.getCanvas("Negative Tracks by
Sector").draw(dataGroups.getItem(9).getH1F("hi_vz_neg_sec_6"));

        canvasTabbed.getCanvas("Negative Tracks by
Sector").draw(dataGroups.getItem(9).getH1F("hi_vz_neg_sec_6_cut"),"same");

        canvasTabbed.getCanvas("Negative Tracks by
Sector").draw(dataGroups.getItem(9).getF1D("f1_vz_neg_sec_6"),"same");


        canvasTabbed.getCanvas("Vertex Comparison").cd(0);

        canvasTabbed.getCanvas("Vertex
Comparison").draw(dataGroups.getItem(10).getH2F("vertex_CVT_TBT_pos"));

        canvasTabbed.getCanvas("Vertex Comparison").cd(1);

        canvasTabbed.getCanvas("Vertex
Comparison").draw(dataGroups.getItem(10).getH1F("vertex_CVT_TBT_diff_pos"),"same");
```

```
        canvasTabbed.getCanvas("Vertex Comparison").cd(2);

        canvasTabbed.getCanvas("Vertex
Comparison").draw(dataGroups.getItem(10).getH2F("vertex_CVT_TBT_neg"));

        canvasTabbed.getCanvas("Vertex Comparison").cd(3);

        canvasTabbed.getCanvas("Vertex
Comparison").draw(dataGroups.getItem(10).getH1F("vertex_CVT_TBT_diff_neg"),"same");


        canvasTabbed.getCanvas("Negative Tracks by Sector Main").cd(0);

        canvasTabbed.getCanvas("Negative Tracks by Sector
Main").draw(dataGroups.getItem(11).getF1D("f1_vz_neg_sec_1"),"same");

        canvasTabbed.getCanvas("Negative Tracks by Sector
Main").draw(dataGroups.getItem(11).getF1D("f1_vz_neg_sec_2"),"same");

        canvasTabbed.getCanvas("Negative Tracks by Sector
Main").draw(dataGroups.getItem(11).getF1D("f1_vz_neg_sec_3"),"same");

        canvasTabbed.getCanvas("Negative Tracks by Sector
Main").draw(dataGroups.getItem(11).getF1D("f1_vz_neg_sec_4"),"same");

        canvasTabbed.getCanvas("Negative Tracks by Sector
Main").draw(dataGroups.getItem(11).getF1D("f1_vz_neg_sec_5"),"same");

        canvasTabbed.getCanvas("Negative Tracks by Sector
Main").draw(dataGroups.getItem(11).getF1D("f1_vz_neg_sec_6"),"same");


        canvasTabbed.getCanvas("Positive Tracks by Sector Main").cd(0);

        canvasTabbed.getCanvas("Positive Tracks by Sector
Main").draw(dataGroups.getItem(12).getF1D("f1_vz_pos_sec_1"),"same");

        canvasTabbed.getCanvas("Positive Tracks by Sector
Main").draw(dataGroups.getItem(12).getF1D("f1_vz_pos_sec_2"),"same");

        canvasTabbed.getCanvas("Positive Tracks by Sector
Main").draw(dataGroups.getItem(12).getF1D("f1_vz_pos_sec_3"),"same");

        canvasTabbed.getCanvas("Positive Tracks by Sector
Main").draw(dataGroups.getItem(12).getF1D("f1_vz_pos_sec_4"),"same");

        canvasTabbed.getCanvas("Positive Tracks by Sector
Main").draw(dataGroups.getItem(12).getF1D("f1_vz_pos_sec_5"),"same");

        canvasTabbed.getCanvas("Positive Tracks by Sector
Main").draw(dataGroups.getItem(12).getF1D("f1_vz_pos_sec_6"),"same");
```

```
    canvasTabbed.getCanvas("beta").cd(0);
    canvasTabbed.getCanvas("beta").draw(dataGroups.getItem(13).getH2F("p_vs_beta_pos"));
    canvasTabbed.getCanvas("beta").cd(1);
    canvasTabbed.getCanvas("beta").draw(dataGroups.getItem(13).getH2F("p_vs_beta_neg"));
    canvasTabbed.getCanvas("beta").cd(2);
    canvasTabbed.getCanvas("beta").draw(dataGroups.getItem(13).getH2F("p_vs_beta_cvt_pos"));
    canvasTabbed.getCanvas("beta").cd(3);
    canvasTabbed.getCanvas("beta").draw(dataGroups.getItem(13).getH2F("p_vs_beta_cvt_neg"));
    canvasTabbed.getCanvas("beta").cd(4);
    canvasTabbed.getCanvas("beta").draw(dataGroups.getItem(13).getH1F("pos_particle_count"));
    canvasTabbed.getCanvas("beta").cd(5);
    canvasTabbed.getCanvas("beta").draw(dataGroups.getItem(13).getH1F("neg_particle_count"));
    canvasTabbed.getCanvas("beta").cd(6);
    canvasTabbed.getCanvas("beta").draw(dataGroups.getItem(13).getH1F("pos_particle_cvt_count"));
    canvasTabbed.getCanvas("beta").cd(7);
    canvasTabbed.getCanvas("beta").draw(dataGroups.getItem(13).getH1F("neg_particle_cvt_count"));

    canvasTabbed.getCanvas("Calculated Beta").cd(0);
    canvasTabbed.getCanvas("Calculated
Beta").draw(dataGroups.getItem(14).getH2F("p_vs_Beta_neg"));
    canvasTabbed.getCanvas("Calculated Beta").cd(1);
    canvasTabbed.getCanvas("Calculated
Beta").draw(dataGroups.getItem(14).getH2F("p_vs_Beta_pos"));
    canvasTabbed.getCanvas("Calculated Beta").cd(2);
    canvasTabbed.getCanvas("Calculated Beta").draw(dataGroups.getItem(14).getH1F("v_time_neg"));
    canvasTabbed.getCanvas("Calculated
Beta").draw(dataGroups.getItem(14).getH1F("v_time_corr_neg"),"same");

    canvasTabbed.getCanvas("Negative Tracks").update();
```

```
        canvasTabbed.getCanvas("Positive Tracks").update();
        canvasTabbed.getCanvas("Monte Carlo").update();
        canvasTabbed.getCanvas("Vertex").update();
        canvasTabbed.getCanvas("CVT Negative Tracks").update();
        canvasTabbed.getCanvas("CVT Positive Tracks").update();
        canvasTabbed.getCanvas("CVT Vertex").update();
        canvasTabbed.getCanvas("Positive Tracks by Sector").update();
        canvasTabbed.getCanvas("Negative Tracks by Sector").update();
        canvasTabbed.getCanvas("Vertex Comparison").update();
        canvasTabbed.getCanvas("Negative Tracks by Sector Main").update();
        canvasTabbed.getCanvas("Positive Tracks by Sector Main").update();
        canvasTabbed.getCanvas("beta").update();
        canvasTabbed.getCanvas("Calculated Beta").update();
    }

    //analyze() performs fitting algorithms fitVertex() to data specified in processEvent() with a
momentum cut, fitMC(), and fitMCSlice().

    private void analyze() {
        //fitting negative tracks vertex
        this.fitVertex(dataGroups.getItem(1).getH1F("hi_vz_neg_cut"),
dataGroups.getItem(1).getF1D("f1_vz_neg"));
        //fitting positive tracks vertex
        this.fitVertex(dataGroups.getItem(2).getH1F("hi_vz_pos_cut"),
dataGroups.getItem(2).getF1D("f1_vz_pos"));
        this.fitVertex(dataGroups.getItem(6).getH1F("hi_vz_pos_cvt"),
dataGroups.getItem(6).getF1D("f1_vz_pos_cvt"));
        this.fitVertex(dataGroups.getItem(9).getH1F("hi_vz_neg_sec_1_cut"),
dataGroups.getItem(9).getF1D("f1_vz_neg_sec_1"));
        this.fitVertex(dataGroups.getItem(8).getH1F("hi_vz_pos_sec_1_cut"),
dataGroups.getItem(8).getF1D("f1_vz_pos_sec_1"));
```

```
        this.fitVertex(dataGroups.getItem(9).getH1F("hi_vz_neg_sec_2_cut"),
dataGroups.getItem(9).getF1D("f1_vz_neg_sec_2"));

        this.fitVertex(dataGroups.getItem(8).getH1F("hi_vz_pos_sec_2_cut"),
dataGroups.getItem(8).getF1D("f1_vz_pos_sec_2"));

        this.fitVertex(dataGroups.getItem(9).getH1F("hi_vz_neg_sec_3_cut"),
dataGroups.getItem(9).getF1D("f1_vz_neg_sec_3"));

        this.fitVertex(dataGroups.getItem(8).getH1F("hi_vz_pos_sec_3_cut"),
dataGroups.getItem(8).getF1D("f1_vz_pos_sec_3"));

        this.fitVertex(dataGroups.getItem(9).getH1F("hi_vz_neg_sec_4_cut"),
dataGroups.getItem(9).getF1D("f1_vz_neg_sec_4"));

        this.fitVertex(dataGroups.getItem(8).getH1F("hi_vz_pos_sec_4_cut"),
dataGroups.getItem(8).getF1D("f1_vz_pos_sec_4"));

        this.fitVertex(dataGroups.getItem(9).getH1F("hi_vz_neg_sec_5_cut"),
dataGroups.getItem(9).getF1D("f1_vz_neg_sec_5"));

        this.fitVertex(dataGroups.getItem(8).getH1F("hi_vz_pos_sec_5_cut"),
dataGroups.getItem(8).getF1D("f1_vz_pos_sec_5"));

        this.fitVertex(dataGroups.getItem(9).getH1F("hi_vz_neg_sec_6_cut"),
dataGroups.getItem(9).getF1D("f1_vz_neg_sec_6"));

        this.fitVertex(dataGroups.getItem(8).getH1F("hi_vz_pos_sec_6_cut"),
dataGroups.getItem(8).getF1D("f1_vz_pos_sec_6"));

        this.fitVertex2(dataGroups.getItem(2).getH1F("hi_vz_pos_cut"),
dataGroups.getItem(6).getH1F("hi_vz_pos_cvt"), dataGroups.getItem(2).getF1D("f1_vz_pos"));



        // fitting MC comparisons

        this.fitMC(dataGroups.getItem(3).getH1F("hi_dp_pos"),
dataGroups.getItem(3).getF1D("f1_dp_pos"));

        this.fitMC(dataGroups.getItem(3).getH1F("hi_dtheta_pos"),
dataGroups.getItem(3).getF1D("f1_dtheta_pos"));

        this.fitMC(dataGroups.getItem(3).getH1F("hi_dphi_pos"),
dataGroups.getItem(3).getF1D("f1_dphi_pos"));

        this.fitMC(dataGroups.getItem(3).getH1F("hi_dvz_pos"),
dataGroups.getItem(3).getF1D("f1_dvz_pos"));
```

```
      this.fitMC(dataGroups.getItem(3).getH1F("hi_dp_neg"),
dataGroups.getItem(3).getF1D("f1_dp_neg"));

      this.fitMC(dataGroups.getItem(3).getH1F("hi_dtheta_neg"),
dataGroups.getItem(3).getF1D("f1_dtheta_neg"));

      this.fitMC(dataGroups.getItem(3).getH1F("hi_dphi_neg"),
dataGroups.getItem(3).getF1D("f1_dphi_neg"));

      this.fitMC(dataGroups.getItem(3).getH1F("hi_dvz_neg"),
dataGroups.getItem(3).getF1D("f1_dvz_neg"));


this.fitMCSlice(dataGroups.getItem(3).getH2F("hi_dp_p_neg"),dataGroups.getItem(3).getGraph("gr_dp_
p_neg"));


this.fitMCSlice(dataGroups.getItem(3).getH2F("hi_dp_theta_neg"),dataGroups.getItem(3).getGraph("gr
_dp_theta_neg"));


this.fitMCSlice(dataGroups.getItem(3).getH2F("hi_dp_phi_neg"),dataGroups.getItem(3).getGraph("gr_d
p_phi_neg"));
   }


   //fitVertex() is the vertex fitting algorithm, and provides the mean, amplitude, and standard
deviations of histograms


   private void fitVertex(H1F hivz, F1D f1vz) {
      double mean  = hivz.getDataX(hivz.getMaximumBin());
      double amp   = hivz.getBinContent(hivz.getMaximumBin());
      double sigma = 1.;
      if(hivz.getEntries()>500) { // first fits
         sigma = Math.abs(f1vz.getParameter(2));
      }
      f1vz.setParameter(0, amp);
      f1vz.setParameter(1, mean);
      f1vz.setParameter(2, sigma);
      f1vz.setRange(mean-2.*sigma,mean+2.*sigma);
```

```
    DataFitter.fit(f1vz, hivz, "Q"); //No options uses error for sigma
    hivz.setFunction(null);
  }
  private void fitVertex2(H1F hivz, H1F hivz2, F1D f1vz) {
    double mean  = hivz.getDataX(hivz.getMaximumBin())-hivz2.getDataX(hivz.getMaximumBin());
    double amp   = hivz.getBinContent(hivz.getMaximumBin())-
hivz2.getBinContent(hivz.getMaximumBin());
    double sigma = 1.;
    if(hivz.getEntries()>500) { // first fits
      sigma = Math.abs(f1vz.getParameter(2));
    }
    f1vz.setParameter(0, amp);
    f1vz.setParameter(1, mean);
    f1vz.setParameter(2, sigma);
    f1vz.setRange(mean-2.*sigma,mean+2.*sigma);
    DataFitter.fit(f1vz, hivz, "Q"); //No options uses error for sigma
    hivz.setFunction(null);
  }

  //fitMC() is is MC data fitting algorithm

  private void fitMC(H1F himc, F1D f1mc) {
    double mean  = himc.getDataX(himc.getMaximumBin());
    double amp   = himc.getBinContent(himc.getMaximumBin());
    double sigma = himc.getRMS()/2;
    f1mc.setParameter(0, amp);
    f1mc.setParameter(1, mean);
    f1mc.setParameter(2, sigma);
    f1mc.setRange(mean-2.*sigma,mean+2.*sigma);
```

```
        DataFitter.fit(f1mc, himc, "Q"); //No options uses error for sigma
        sigma = Math.abs(f1mc.getParameter(2));
        f1mc.setRange(mean-2.*sigma,mean+2.*sigma);
        DataFitter.fit(f1mc, himc, "Q"); //No options uses error for sigma
        himc.setFunction(null);
    }
    private void fitMCSlice(H2F himc, GraphErrors grmc) {
        grmc.reset();
        ArrayList<H1F> hslice = himc.getSlicesX();
        for(int i=0; i<hslice.size(); i++) {
            double  x = himc.getXAxis().getBinCenter(i);
            double ex = 0;
            double  y = hslice.get(i).getRMS();
            double ey = 0;
            double mean  = hslice.get(i).getDataX(hslice.get(i).getMaximumBin());
            double amp   = hslice.get(i).getBinContent(hslice.get(i).getMaximumBin());
            double sigma = hslice.get(i).getRMS()/2;
            F1D f1 = new F1D("f1slice","[amp]*gaus(x,[mean],[sigma])", hslice.get(i).getDataX(0),
hslice.get(i).getDataX(hslice.get(i).getDataSize(1)-1));
            f1.setParameter(0, amp);
            f1.setParameter(1, mean);
            f1.setParameter(2, sigma);
            f1.setRange(mean-2.*sigma,mean+2.*sigma);
            DataFitter.fit(f1, hslice.get(i), "Q"); //No options uses error for sigma
            if(amp>50) grmc.addPoint(x, f1.getParameter(2), ex, f1.parameter(2).error());
        }
    }

    //setAnalysisTabNames() sets the Analysis tab names specified in main()
```

```
    public void setAnalysisTabNames(String... names) {
        for(String name : names) {
            canvasTabNames.add(name);
        }
        canvasTabbed = new EmbeddedCanvasTabbed(names);
    }

    //printCanvas() produces the plots

    public void printCanvas(String dir, String name) {
        // print canvas to files
        for(int tab=0; tab<canvasTabNames.size(); tab++) {
            String fileName = dir + "/" + this.analysisName + "_" + name + "." + tab + ".png";
            System.out.println(fileName);
            canvasTabbed.getCanvas(canvasTabNames.get(tab)).save(fileName);
        }
    }
    public boolean testMCpart(Particle mc, Particle rec) {
        if(Math.abs(mc.px()-rec.px())<2.5 &&
           Math.abs(mc.py()-rec.py())<2.5 &&
           Math.abs(mc.pz()-rec.pz())<2.5) return true;
        else return false;
    }
}
```