Geane Track Fitting

Nicholas Kinnaird
nickkinn@bu.edu

September 1, 2017

Abstract

In this document I will detail the concepts and mathematics of GEANE track fitting, as well as its implementation in the gm2 simulation framework for the Muon g-2 Experiment at Fermilab. This is done both for documentation purposes as well as a precursor to sections of my future thesis. This document assumes some familiarity with art on part of the reader, though such knowledge is not necessary for many sections. Note that the tracking code and framework is in constant development, so the details contained within may be outdated at the time of reading. I've tried to be thorough in my descriptions, but it is possible that some finer points have been left out of this document.

Contents

1 Introduction

The Muon g-2 Experiment at Fermilab uses straw tracking detectors to measure decay positron trajectories for the purpose of determining the muon beam distribution and its characteristics. By fitting these tracks and extrapolating back to the average decay point, the beam can be characterized in a non-destructive fashion. This is important because of the need for matching the average observed magnetic field of the decaying muons and their resulting decay positron directions which result in the ω_a frequency, as seen in

$$\vec{\omega}_a = \frac{e}{m}[a_\mu \vec{B} - a_\mu(\frac{\gamma}{\gamma+1})(\vec{\beta}\cdot\vec{B})\vec{B} - (a_\mu - \frac{1}{\gamma^2-1})(\vec{\beta}\times\vec{E})]. \quad (1)$$

The trackers are also useful for determining general beam diagnostics as well as the pitch correction and to a lesser extent the electric field correction, terms 2 and 3 in Equation 1 respectively. The tracking analysis can be done independently of, or in tandem with, the calorimeters. Cross-checking separately for pileup removal, hit verification, etc. is a powerful tool. Combining them in order to provide the muon distribution that the calorimeters directly see for the ω_a calculation is perhaps the most important role of the tracker. (An EDM analysis needs to be done separately.) It is worth noting that there is a large percentage of tracks hitting the calorimeters that hit zero or only a small number of tracking modules, which this Geane fitting code is not capable of handling. With three trackers, approximately 5% of decaying muons will result in measureable positron tracks assuming no pileup in the tracker, many of which do not hit the nearest calorimeter. Note that the integration of the two detector systems in the code (tracker-calo matching) has just recently been initiated, DocDB 7514.

Each tracker module consists of 4 layers of 32 straws with a stereo angle of 7.5 degrees, the first two "U" layers oriented with the tops of the straws at a greater radial position, and the second two "V" layers oriented with the bottoms of the straws at a greater radial position. A tracking module is shown in Figure 1. There are 3 tracker stations located at the 0th, 12th, and 18th sections of the ring, counting clockwise from the top most point of the ring where the inflector resides. Figure 12 shows this. (Station 18 was installed for the commissioning run, with station 0 planned for the fall. Station 12 might or might not be installed sometime in the future.) Each station consists of 8 tracking modules arranged in a staircase pattern that follows the curvature of the ring as seen in Figure 2. Further hardware and electronics information regarding the trackers will be omitted in this document.

Because of the proximity of the trackers to the muon beam, they will lie within a region of varying magnetic field. The radial field of the trackers rises from 0 Tesla at the outer ends to roughly .3 Tesla at the inner top and bottom ends, and the vertical field drops approximately 50% from the storage dipole field of 1.451 Tesla. Shown in Figures 3 and 4 is the location of the tracker with respect to the horizontal and vertical fields respectively. These large field gradients over the tracking detector region and the long extrapolation distance back to the muon decay point are special to Muon g-2. This is one of the main motivations for using the Geane (Geometry and Error Propagation) fitting algorithm and routines, which has direct access to the field.

Figure 1: Shown is a picture of one of the many tracking modules used in the Muon g-2 experiment. The first layer of straws with a stereo angle of 7.5 degrees can be seen, with the other 3 straw layers hiding behind it. The beam direction is roughly into the page in this picture, to the left of the end of the module, and this view is what the decay positrons will see. Picture provided by James Mott.

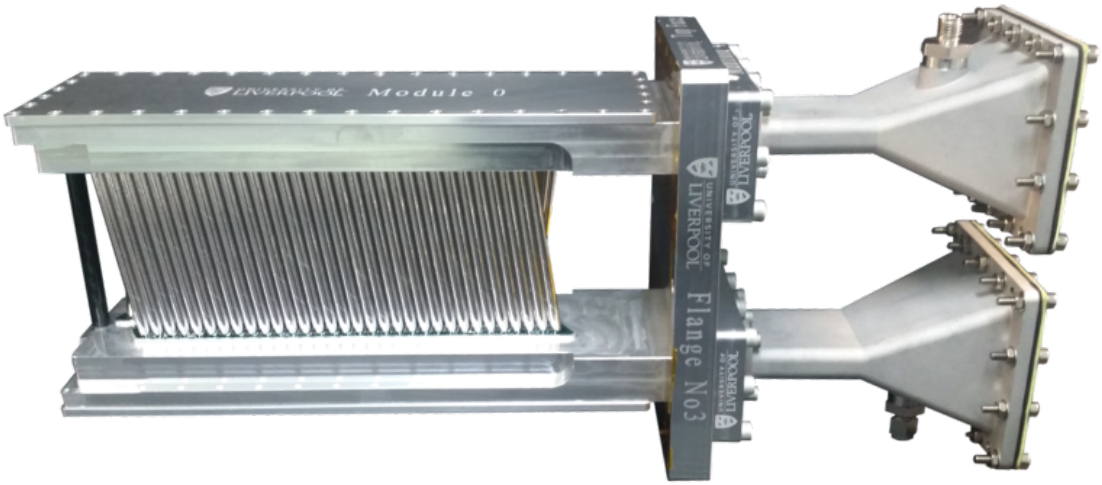


Figure 2: Tracker modules are arranged in the shown staircase pattern. In green and dark blue is the edge of the vacuum chamber (where the dark blue identifies the modification that was made to the old vacuum chambers), and it can be seen that vacuum chamber walls lie at the ends of the outside tracking modules. The position of a calorimeter can be seen in cyan at the right. The dark red spots are the locations of the outside magnet pole tips. From the shown geometry one can see that many positrons will hit either the tracker or the calorimeter but not both due to the acceptance differences.

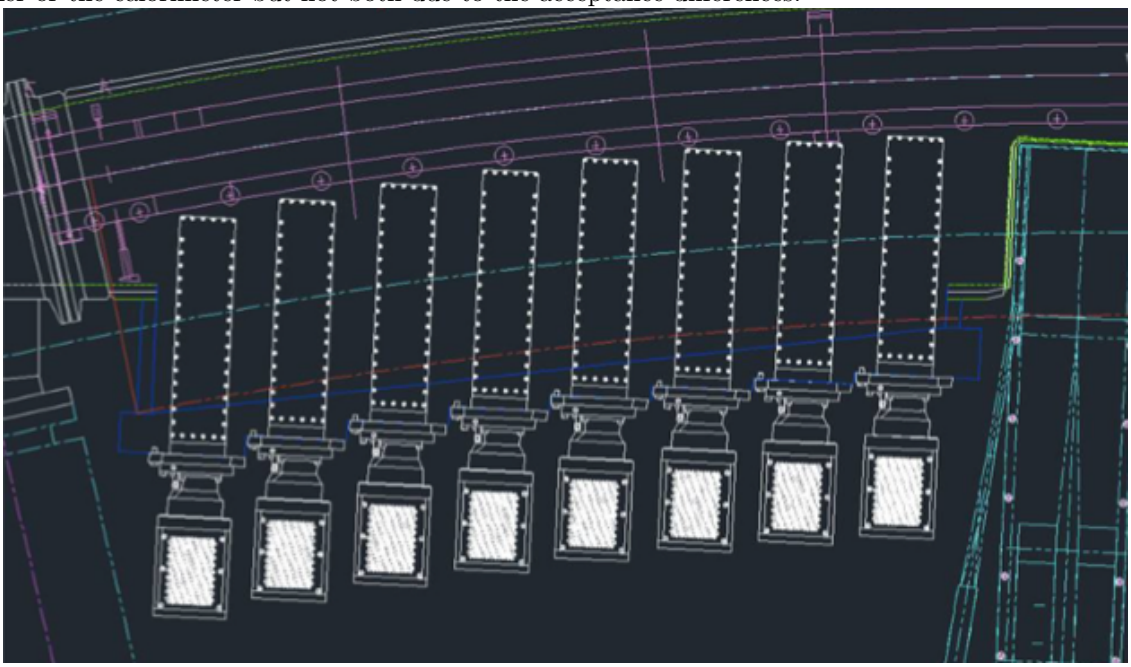


Figure 3: Shown is the vertical field of the g-2 magnet in and around the storage region as calculated in Opera 2D. The center of the storage region lies at 7.112 m along the x axis. The black box shows the rough location of the tracker with respect to the field (size exaggerated slightly). It can be seen that there is a large inhomogeneity within the tracker space, goring from left to right.

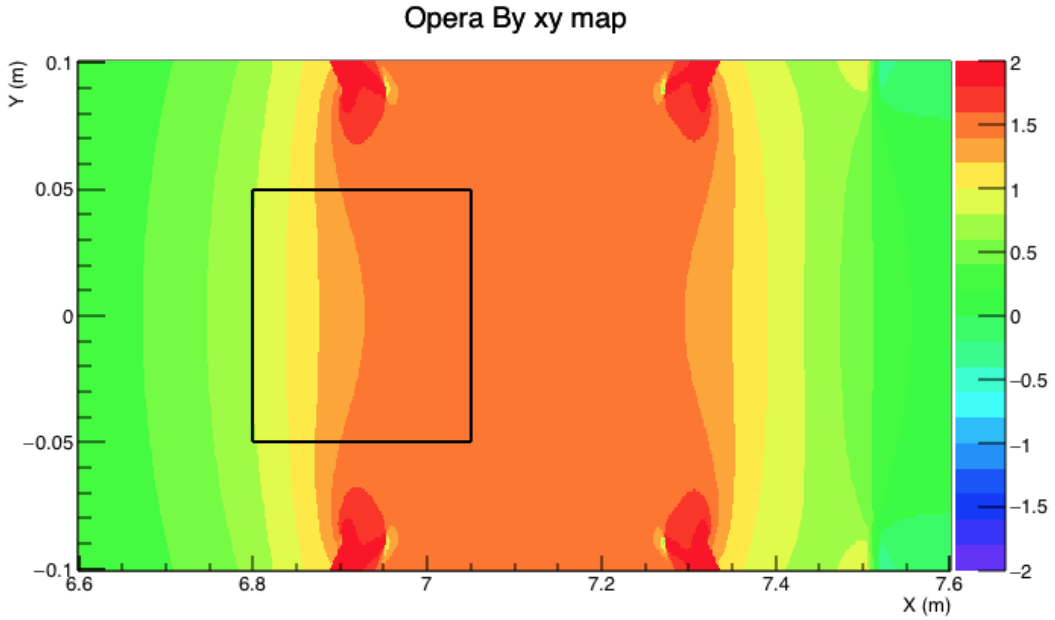
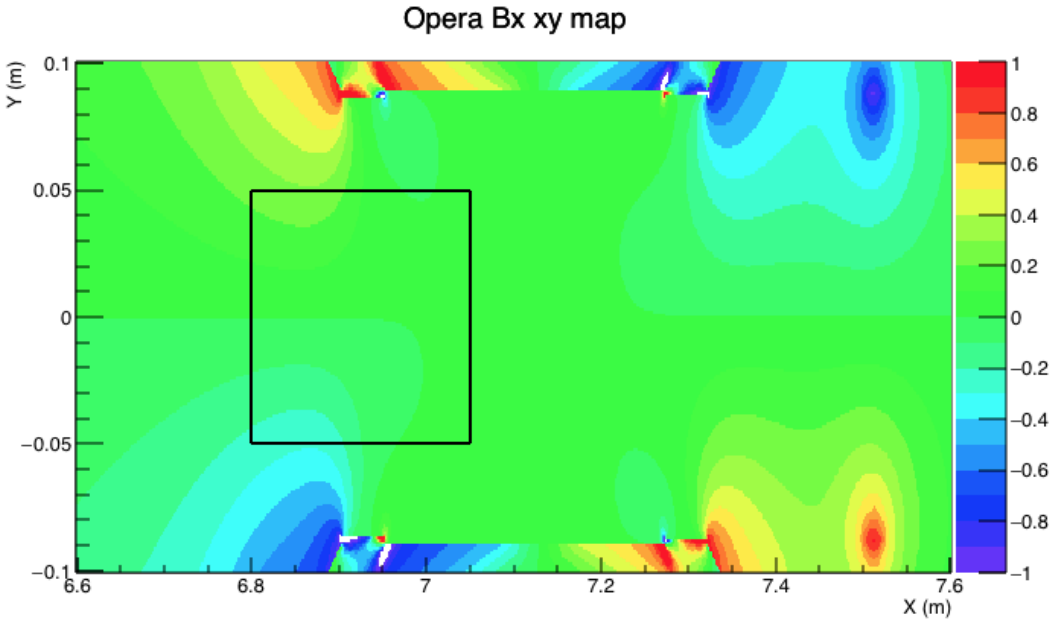


Figure 4: Shown is the radial field of the g-2 magnet in and around the storage region as calculated in Opera 2D. The center of the storage region lies at 7.112 m along the x axis. The black box shows the rough location of the tracker with respect to the field (size exaggerated slightly). It can be seen that there is a large homogeneity at the inner upper and lower ends compared to the right center. The shape of the pole pieces and tips can readily be seen.



The Geane fitting routines originated in Fortran with the EMC collaboration, and was used in the precursor E821 experiment as well as the PANDA experiment with some success [1], [2]. (I'm not actually aware of a useful reference for it's use in E821, and there are some other instances of its use as well in other experiments. In E821 there was a single tracking chamber which was never put to full use.) The core error propagation routines were at some point added to Geant4 under the error_propagation directory which is included in all default installs. The tracking code strengths lie with its direct implementation and access to the Geant4 geometry and field, and its ability to handle the field inhomogeneties. The Geane fitting algorithm code which makes use of the Geant4 error propagation routines follows the structure of [1] and is detailed in the Formalism section in this paper. It is a relatively straight forward least squares global χ^2 minimization algorithm.

2 Formalism

I recommend reading [1], Chapter 4 of [2], and [3] in order to best understand the fitting algorithm. However, due to the at times confusing notation, ommitted equations or concepts, and differences between papers, I have attempted to summarize here the different sources and present the material in a more understandable and readable format. The implementation of the fitting algorithm into the code follows this section.

One can define a χ^2 for a track in the usual way by dividing the residuals of measured and predicted track parameters by their errors:

$$\chi^2 = (\vec{p} - \vec{x})^T (\sigma^{-1})(\vec{p} - \vec{x}), \tag{2}$$

where $\vec{p}$ are predicted track parameters from a fit to the measured track parameters $\vec{x}$, and σ is a covariance matrix of errors on the fitted parameters. The Geant4 error propagation routines can be used to determine these predicted parameters and error matrices by propagating track parameters from some initial guesses. By minimizing this χ^2 with respect to the track parameters one can then fit and improve the track. The Geant4 error propagation routines propagate particles along their average trajectories neglecting the effects of discrete processes, using a helix equation along small enough steps where the change in the magnetic field is small. The predicted parameters are then a function of path length:

$$p_l = F_{l,l_0}(p_0), \tag{3}$$

where the path length can be defined how one wishes. In our system we have tracker planes defined at X positions, and limit path lengths to reach those planes. (From here on the dependence on path length or X position will be neglected, in favor of using plane indices.) In tandem, error matrices describing the expected distribution in true parameters about those predicted parameters due to said discrete process are also calculated:

$$\sigma^{ij} = < p^i p^j > - < p^i > \cdot < p^j >, \tag{4}$$

where i and j are track parameter indices. These parameter vectors are 5x1 objects defined in some track representation, as described in the Coordinate Systems section. The propagation of these parameters and error matrices are done using transport matrices, which express the infinitesimal changes in parameters at some plane (or path length) with respect to the parameters at some previous plane (or previous path length):

$$\delta p_N = T_{N,N-1} \delta p_{N-1}, \tag{5}$$

$$\sigma_N = T_{N,N-1} \sigma_{N-1} T^T_{N,N-1}. \tag{6}$$

Said transport and error matrices are 5x5 objects since the parameter vectors are 5x1 objects as described above. The calculation of these transport matrices, as well as details on the functional form of 3 are shown in [4].

With parameters defined on such planes, one can define the χ^2 as:

$$\chi^2 = \sum_{i=1}^{N} [(p_i(p) - x_i)^T (\sigma_i^{-1})(p_i(p) - x_i)], \tag{7}$$

where p_i are the average predicted parameters from some general starting parameters p. At first order one can solely include the measurement errors on parameters, which fill in the diagonals of σ_i, if random processes can be neglected. Unmeasured parameters should have measurement errors of infinity (or some large value) along the diagonals in the code, which account for the fact that residuals for unmeasured parameters do not exist. When the error matrix is inverted all rows and columns of the matrix with these large numbers will fall to 0 in the χ^2.

In order to get the best fit track, the χ^2 should be minimized with respect to the initial track parameters p, and evaluated at some chosen or fitted parameters:

$$\frac{\partial \chi^2}{\partial p}|_{p=p'_0} = 0, \tag{8}$$

resulting in

$$\begin{aligned} 0 = \sum_{i=1}^{N} [(\frac{\partial p_i(p)}{\partial p}|_{p=p'_0})^T (\sigma_i^{-1})(p_i(p'_0) - x_i) \\ +(p_i(p'_0) - x_i)^T \frac{\partial(\sigma_i^{-1})}{\partial p}|_{p=p'_0}(p_i(p'_0) - x_i) \\ +(p_i(p'_0) - x_i)^T (\sigma_i^{-1})(\frac{\partial p_i(p)}{\partial p}|_{p=p'_0})] \end{aligned} \tag{9}$$

where the 1st and 3rd terms are identical, and the 2nd term is small if one assumes that the error matrix doesn't change much with respect to the starting parameters. (Fair since most of the error comes from measurement, and as long as the initial guess is decent enough such that the path length through material doesn't change appreciably from one iteration to the next.) This simplifies to:

$$\sum_{i=1}^{N} T_{i0}^T (\sigma_i^{-1})(p_i(p'_0) - x_i) = 0, \tag{10}$$

which is just the top term with

$$T_{i0} = \frac{\partial p_i(p)}{\partial p}. \tag{11}$$

To solve this make the substitution

$$p_i(p'_0) = p_i(p_0) + \frac{\partial p_i(p_0)}{\partial p}\Delta p_0 = p_i(p_0) + T_{i0}\Delta p_0, \tag{12}$$

where p'_0 are the improved starting parameters for the next iteration calculated from the previous starting parameters p_0, and Δp_0 are the changes in the starting parameters to improve the track. This equation can be plugged into the above if one makes the assumption that T_{i0} does not change much from one iteration the next, which follows from the inherent nature of making small adjustments to the track in order to improve it.

After simplifying one arrives at

$$\Delta p_0 = \sigma_{p_0} \sum_{i=1}^{N} T_{i0}^T (\sigma_i^{-1})(x_i - p_i(p_0)), \tag{13}$$

where

$$\sigma_{p_0} = [\sum_{i=1}^{N} T_{i0}^T (\sigma_i^{-1}) T_{i0}]^{-1}, \tag{14}$$

is the 5x5 covariance matrix of fitted parameters on the starting plane, whose diagonals describe the errors in the 5 track parameters on that plane and in the region close to it. (The fit does not directly return fit

errors for track parameters on other planes.) Δp_0 along with χ^2 is exactly what we want to determine since that is what allows us to fit and improve the track from iteration to iteration.

However, since random processes should not be neglected for optimal tracking results, it makes more sense to return to the original χ^2 in equation 2, only now the included matrix and vector objects are combined into one large linear algebra equation. Instead of a sum over N 5x1 objects multiplying 5x5 error matrices, the vectors are combined into a single 5Nx1 vector multiplying a single 5Nx5N matrix. The 5x5 diagonal blocks of this large error matrix should now include the effects due to material processes as calculated in Geant from equation 4 as well as the measurement errors.

Because now parameters at one plane are no longer independent of the parameters at other planes, due to correlations from these random processes, it's necessary to add off-diagonal elements into the large error matrix. These 5x5 blocks come from

$$\sigma_{MN} = T_{MN}\sigma_N, \tag{15}$$

for the top diagonals, and the transpose for the bottom diagaonals, where M and N are two separate planes within the detector. (σ_N is the error matrix on plane N calculated from the starting plane.) This follows from equation 4 evaluated at plane M with respect to a path length from plane N, and not plane 0, which is equivalent to 15.

You can then minimize the χ^2 in the same way, only again with the matrix objects being aggregates of the per plane objects:

$$\Delta\vec{p}_0 = \sigma_{p_0}\tau^T\sigma^{-1}(\vec{x} - \vec{p}), \tag{16}$$

$$\sigma_{p_0} = [\tau^T\sigma^{-1}\tau]^{-1}, \tag{17}$$

where τ is the combined transport matrices from the individual 5x5 matrices, a 5Nx5 object.

The unmeasured parameter errors of infinity still come into play in the final calculation in the same was as before. Because however these matrix objects are very large, and the tracking must have a certain amount of speed in order to keep up with data, it is useful to reduce the size of these matrices. (It also makes things easier programming wise. Note that there are other some other ways to speed things up, specifically the banded inversion method as described in reference [3]. This method was not used in favor of getting the code working in the simpler form in the first place, but it is a possiblity in the future to use this technique to speed things up even more.) It suffices to simply remove all rows and columns where said infinity values exist in the error matrix. This is mathematically equivalent to inverting the error matrix with the infinities included, which make all rows and columns where they exist go to zero. The associated unmeasured parameter rows in the residual vector and transport matrices must similarly be removed. This results in an Nx1 residual vector, NxN error matrix, 5xN combined transport matrix transpose, which multiply against the 5x5 covariance matrix out front to still result in a 5x1 fix to the starting parameters, and a scalar χ^2 value. (Note that these element removals should be done just before the final calculation, and not higher up in the algrebra, otherwise plane correlations are not properly calculated.)

By calculating the last two equations one can fit the track, acquire a χ^2 describing the degree of the fit, determine how the track parameters can be improved at the starting point, and calculate errors on those starting parameters. This algorithm can be iterated a number of times to get a best fit track until successive iterations produce no improvement, where usually 3 or 4 iterations is enough. Note that there is remarkable robustness with respect to the initial starting parameters in fitting the track. Of course if the initial starting paramaters are too poor, then the fit will not converge. All of these calculations are completed within the GeaneFitter.cc file within the framework.

3 In the gm2 Framework

Depending on the level to which a user wants to utilize or develop the tracking code, they may choose which branches of the gm2 framework to checkout. If just fitting tracks and viewing plots, then just gm2tracker on the develop branch may be checked out. If doing further development, it can be a good idea to checkout the artg4, gm2dataproducts, gm2geom, gm2ringsim, gm2tracker, and gm2utils packages of the gm2 repository, and be on the feature/trackDevelop branch in all instances. Every so often these branches are merged with the main develop branches, where the tracking code will work but might not have the latest code changes.

3.1 Event Generation, Geometry, and Material

While not a direct part of the reconstruction and fitting code, understanding the event generation in the simulation can be useful, or necessary if developing. Here is given just a quick summary of relevant information. Event generation before fitting is done using the mdc fcl files in gm2ringsim using the main simulation, where one can include the tracker dummy plane geometry for truth comparison with the Geane fitting if so desired. These are built in the TrackerDummyPlane_service and associated geometry files, and are included in the common mdc fcl files. (Without these perfectly placed dummy planes it is very difficult to gather truth Monte Carlo data and even small inconsistencies can be seen in the tracking results.) If one wishes there are also a couple of modified mdc#-geane fcl files available for use with reduced geometry. For stats reasons it is usually a good idea to include all 3 trackers when generating events.

Fcl parameters exist for the StrawTrackerCadMesh_service and Straws_service in order to turn material on or off, "materialTracker" and "strawMaterial" respectively. The rest of the geometry has to be manually changed and rebuilt in order to remove material if one wishes. These include the services VacuumChamberCadMesh and the World, as well as the "buildSupportPost" option in strawtracker.fcl, "buildTrolly" in vac.fcl (where the associated material is hardcoded in), and "trolleySupportMaterial," also in vac.fcl. One should make sure to perform reconstruction with the same material parameters as were used in the event generation for proper results. (These options are primarily for debugging the tracking. Near perfect results have been shown for tracking within a vacuum world, DocDB 4876 and DocDB 4894.) The user should also make sure to perform the reconstruction with the same magnetic fields as were used in the event generation, though small changes won't make much of a difference.

3.2 Reconstruction Flow

The overall tracking infrastructure and reconstruction flow can be seen in Figure 5. Data coming from simulation or the real experiment are turned into art objects upon which the tracking framework acts. The RunGeane.fcl file performs the entire chain within the blue box in Figure 5, excepting the track extrapolation stage. Within the reconstruction flow objects called Track Candidates are produced, which are the input to the Geane fitting code.

The Geane fitting specific flow can be seen in Figure 6. There is a number of files involved in the production of fitted tracks. Detailed information regarding all of these files is given below, but here is provided a shorter summary. Track candidates from upstream are used as the input to the producer module GeaneReco, which produces TrackArtRecords and TrackStateArtRecords, which are then passed to the extrapolation or refinement stage. GeaneReco also produces GEANEArtRecords, which is the data product that is used and updated throughout the fitting process until the track converges or fails, and may be passed through filters or to analyzers at the end of the fitting. There are a number of utils files that the Geane fitting utilizes. GeaneReco directly links to the GeaneFittingUtils which contains a number of methods for the different fitting modes, as well as the general fitting loop which links to the GeaneFitter class, and iterates until the track fitting succeeds or fails. GeaneFitter provides a χ^2 and improvement to the track as described in the Formalism section. GeaneFittingUtils links to GeaneParamUtils and GeaneLRUtils. GeaneParamUtils deals with any code that modifies parameters regarding the fit, both setting up the initial fit parameters, and calculating the predicted parameters of the fit within Geant4 using the error propagation routines. GeaneLRUtils deals with any code regarding the left and right information of the track for fitting.

Note that after a single iteration, one pass through GeaneFitter and one pass through the relevant fitting utils, a χ^2 and an improvement for the track will be produced, but the track predicted parameters and objects will be based on the previous starting position and momentum guess before it has been updated, and so do not correspond to the fitted track. It's necessary to perform at least a second iteration for this reason, and is why no track converges under strict criteria in under 3 iterations.

Also worth mentioning is that the GEANEArtRecords are passed by reference or pointer throughout the various fitting files, allowing them to be updated along the way in a natural manner. At the same time, many methods within the fitting code return ints describing whether the track fitting has failed at different stages to various reasons, where a value of 0 signifies success at that stage. A returned int with a non-zero value might for example signify that too many digits were included in the initial track candidate, or that Geant is having tracking issues, etc.

Figure 5: Shown is the infrastructure flow for the entire track reconstruction chain. In the green blocks are the sources of track data to fit, either from Geant simulation, or real data. In blue is the offline reconstruction block. Straw digits are formed in the digitalization step, those are then calibrated and grouped into time islands, clusters, and seeds which then combine to form track candidates. It is these track candidates which are the input into the Geane fitting code (and other future fitting code). The fitting code then outputs tracks which the track extrapolation stage will run over. This picture is taken from one of Tammy Walton's talks. Note that there is some iteration between stages that is not shown.

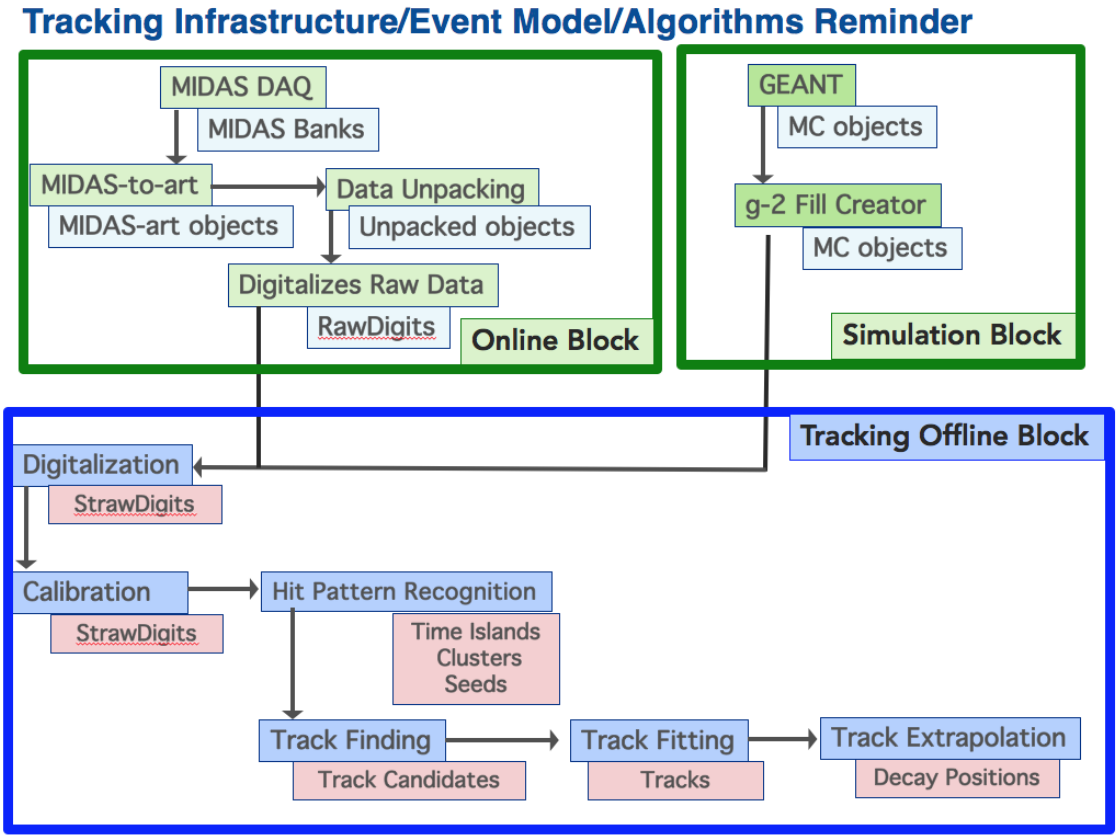
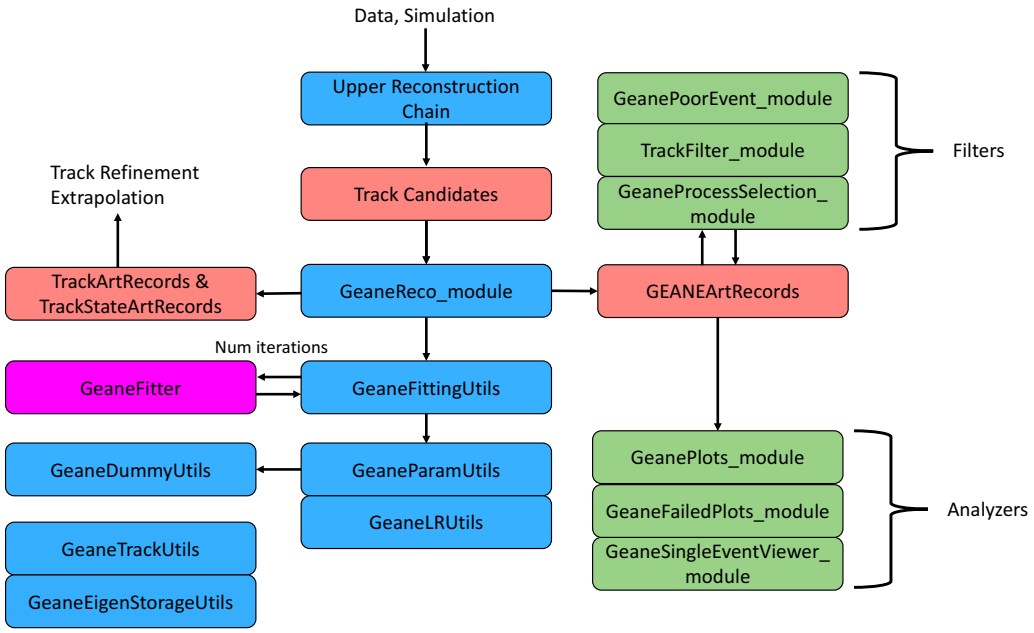


Figure 6: Shown is the Geane fitting code flow. See the text for a thorough explanation of this flow, and the specifics of each box. GEANEArtRecords are created in the GeaneReco producer module, and are passed by pointer and reference to the different utils files which then update them for the different fitting modes and with the fitting results.



3.3 fcl File Specifics - RunGeane.fcl

Reconstruction is performed using RunGeane.fcl in gm2tracker on the events generated from somewhere higher in the chain. (There is also a RunGeane-midasdata.fcl file which is more tuned to running on real data, with certain fcl parameters set appropriately and some modules left out of the chain.) It is important to explain some fcl parameters necessary for the reconstruction. First, there is a fcl parameter "useSD" for the Straws geometry service to turn off the straw sensitive detectors so that in the reconstruction phase hits are not regenerated. If this is not included, it will default to true and cause crashes in the reconstruction. Secondly, the RunGeane.fcl file loads all 3 trackers necessary for symbol and name definitions, but such that tracker 0 is rotated to have tracking planes parallel to the global Geant X axis (using the "rotateArcTracker" fcl parameter for the Arc service). This is done to avoid the issue of error propagation instability close to the Z axis, as detailed in DocDB 4567, while at the same time still observing the correct azimuthally symmetric 2D field for the tracks. Track hits are rotated from their separate tracker frames to this one reconstruction frame with the GeaneWorldTracker[0], GeaneWorldTracker[12], and GeaneWorldTracker[18] transforms defined at the bottom of StrawTrackerCadMesh_service.cc, and as detailed in the Coordinate Systems section. As a reminder there are material fcl parameters for the straws and straw trackers which should be the same as were used in the event generation. The RunGeane.fcl file also includes some analyzer and filter modules, which are detailed further below, and can be turned on and off at will. (Depending on the stage of code development, these are turned off and on in various git commits. For instance with the latest grid commands, it is necessary to turn the analyzers off by default. Also be wary of changing output file names, there is a certain structure they must have with keywords in order to work with the latest grid submission scripts.)

While the pieces of the chain before the track fitting (digitalizing, seeding, clustering, t0 finder, and track finding) are distinct and separate from the majority of the material detailed in this document, it is important to understand those stages to some level and the different options which will affect the tracking results. See a collaboration talk by Tammy, DocDB 5601, for a general overview on this. Here is a short summary, not all method or file calls are detailed. Most of these files and parameters should be left unchanged unless one is familiar with the subject matter. These files are constantly being changed in small but important ways,

so again the information detailed here may be out of date by the time of reading.

1. **Digitalizer_module.cc**
   Creates StrawDigits which will ultimately be combined to form the Track Candidates that the fitting will act over. It is the variables in this art record that are updated and correspond to real data: which straw was hit, the time of the hit, and the dca of the hit. The StrawDigits also contain a pointer to a struct of Monte-Carlo truth data, StrawMCDigit, which can also be accessed for track information. The Digitalizer module has several fcl parameters to change things like the gas drift time, drift model, allowing recorded negative times, and some particle cuts. By default it uses a Gaussian drift model which is called when the Digitalizer calls the DriftTimeModels.cc file, where it will gaussian smear the hit time from the MC digit time. These fcl parameters are located in digitalizerParams.fcl. The smearedDriftTimeSeed parameter should be set to 0 if one wants random results from run to run, which is necessary for grid jobs. The drift time model can also be set to the Garfield version, which will produce more realistic measurement parameters. If this is used the drift time calculator in the DriftDistanceCal module should also be set to the Garfield version. These more realistic hit data result in reconstructed tracks that haven't been fully analyzed yet, with a non-uniform p value distribution.
2. **TimeIsland_module.cc**
   Groups the digits from the Digitalizer in time. Has some options for changing the time window width. fcl parameters are located in recoIslandParams.fcl. There is ongoing work in relation to this module for dealing with real data, where there is a much increased number of hits closer in time. The short summary here does not reflect the difficult and important task that this module has.
3. **ClusterFormation_module.cc**
   Groups neighboring digits in the same view to form clusters. fcl parameters are located in recoClusterParams.fcl.
4. **T0Finder_module.cc**
   Calculates a t0 for the track. Has an option for which model to use for calculating said t0. The default is using truthData if one is running the tracking over simulated hits. If one is looking at real data there are other options. It's important to note that the t0 for a track can be difficult to calculate, and the current goal is to combine information from the calorimeter to get this number. fcl parameters are located in recoT0FindingParams.fcl.
5. **DriftDistanceCal_module.cc**
   Calculates the drift times and distances within the straws based on the calculated t0 from the previous module. Has options for different drift calculators and parameters, including a "useTrueDigitT0" parameter for using mc truth information. The drift time is calculated from the digit hit time (calculated in the Digitalizer) minus the digit t0. If true t0 is not used, then the dca smearing will not be perfectly gaussian. fcl parameters are located in recoClusterParams.fcl.
6. **SeedFormation_module.cc**
   Groups neighboring clusters in the same module to form seeds. There are a couple of fcl parameters for this module directly relevant to the track fitting. The first is "reconstructPosition" which will fill seed nodes which form the seed, and essentially correspond to indidual digits - but which contain some left-right ambiguity information important for track fitting. The second is "useGeometryLR" which uses the straw geometry information to make an initial guess at the left-right choices for doublets within the track. fcl parameters are located in recoSeedParams.fcl.
7. **Aside** It's important to note that there is another iteration of the clustering module which is improved now that better drift times and distances have been calculated from the previous modules, using the same fcl paramters. This will almost certainly change in the future as the tracking code becomes more developed.
8. **TrackFinding_module.cc**
   Groups Time Islands and Straw Digits into Track Candidates based on their positions in space. There are two finders available for use, fcl parameter "trackFinderName", the Simple Track Finder (SimpleTrackFindingUtils.cc) and the Long Track Finder (LongTrackFindingUtils.cc). The former runs solely off the generated time islands with no higher level pattern recognition or track cuts. If this is enabled,

then the second clustering module module can be left out of the reconstruction chain. The latter, Long Track Finder, forms track candidates based on combining seeds and is the default used. It has stricter fcl conditions, such as a default cut of 6 or more planes hit for the track, and some further pattern recognition. Both finders make use of the SimpleCircleFitter.cc class which fits a circle to the U and V digit positions to calculate a starting position and momentum guess for the track, which is currently necessary for the Geane fitting. The Long Track Finder also has a least squares minimizer which is unimportant for the Geane fitting. fcl parameters are located in recoFindingParams.fcl, which then also links to some other fcl files depending on the finder used.

3.4 geaneFitParams.fcl

Because the Geane track fitting is the subject of this document, it will go more into detail of the Geane fitter fcl parameters, located in the geaneFitParams.fcl file. Some of these parameters are very likely to change or disappear in the near future.

1. **module_type**
   The ordinary art producer parameter which tells art which cc file to run, this is always set to GeaneReco unless the filename changes.
2. **G4EVERBOSE**
   This parameter corresponds to the iverbose level present in the Geant4 error propagation files and can be manually set here, from 0 to 5. It is for the debugging of deeper level Geane fitting code. Note that unless Geant is compiled with the flag by the same name, then setting this verbose level will do nothing, because these verbose statements are located within #ifdef blocks within the code. The default is to keep the #ifdef blocks in order for the compiler to ignore them with the flag unset to improve the speed of the tracking. Since we utilize our own slightly modified error propagation files copied into the gm2tracker/gm2Geane directory, it would be necessary to make the relevant changes there as well.
3. **trackingVerbose**
   Geant4 tracking verbose level, from 0 to 5. This is always available but should only be used for debugging.
4. **matrixDebug**
   This boolean paramater turns on and off the extensive matrix debugging output, used for higher level debugging. Note that these outputs are included in the log file and the log file threshold needs to be set to "DEBUG" as well. In the future it might be beneficial for speed reasons to completely remove these debugging outputs. This parameter should always be turned off if looking at a large number of events.
5. **numPassesWireFit, numPassesSeqFit**
   These are parameters used for the left-right sequence checking routines for a fullSeqFit. numPassesWireFit is the number of passes that the track fits to the wire centers, before which the best sequences are passed to the full fit code which iterates a number of times equal to numPassesSeqFit. It was observed in the past that 3 passes for each of these was sufficient for good tracking results. See the Left-Right section.
6. **convergenceCriteria**
   This is a scalar convergence criteria telling whether the track fitting has been successful or not, with a default value of .1. As long as the current iteration χ^2 minus the previous iteration χ^2 is less than this value, the track is considered to have converged. Otherwise it continues iterating until it succeeds, fails, or is manually cut off at some number (currentlt set at 10 iterations).
7. **useCircleGuess**
   This boolean parameter tells the code whether to use the circle fitter results for the track starting guess or not, called in the track finding part of the chain. Its default should remain true unless one wants to return to the uniform starting parameter smearing for debugging purposes.

8. **lockLowDCAs**
   This fcl parameter sets a dca threshold below which measured hits with small dcas have their measured positions set to the wire centers, and their left-right sides locked to the center. This will be necessary when fitting data due to the large measurement uncertainty of hits close to the wire. (It also speeds up the code with less left-right choices to iterate over.) Its default value of 0 corresponds to turning this functionality off, while a positive number corresponds to the threshold radius in mm that dcas less than will be locked.
9. **useNodes**
   This boolean parameter sets whether to use the left-right information from the seed nodes. The recoSeedParams fcl parameter "reconstructPosition" needs to be set to true for this to work, and the parameter "useGeometryLR" should also be set to true, being that it is the only upstream left-right code currently implemented.
10. **fitMode**
    This is the most important fcl parameter for the Geane fitting. It defines what type of fitting mode one would like to use for fitting the incoming track candidates. There are currently 4 separate fitting modes. "truthLRFit" only works with simulated data and will fit to the unambiguous left-right UV values from the straw digits. "wireFit" will fit a track to the wire centers of the incoming digits, with a corresponding uniform error proportional to the gas diameter. "mainFit" will first do a wire fit to the track in order to make an initial guess at the left-right choices for each hit, and will then fit to those left-right choices. (The default main fit actually does a second wire fit in the middle where it first locks left-right choices for doublets where the left-right choice is guaranteed to be known, which improves the tracking results slightly.) Finally there is the "fullSeqFit" which will first do the wire fit, and will then go about checking all left-right sequences before doing full fits on the set of best sequences. (There is also some code to lock the left-right sides based on the doublets.) See the Left-Right section.
11. **useTangentLR**
    This fcl parameter links to code which does a slightly more advanced locking of the doublet left-right geometry include track tangent information, defaulted to false. Written by Joe Price.
12. **rseed, yPosChange, zPosChange, xMomChange, yMomChange, zMomChange**
    These are the uniform smearing bounds for the starting position and momentum of the track from truth, in units of mm and MeV. These are largely unused now since the circle fitter provides a natural level of smearing automatically, but should be kept for future debugging purposes. The rseed parameter is the number seed for the random number generator. The default should be 0 for the root TRandoms unless one wants to reproduce results.

3.5 Reconstruction Files

Here will be given a summary of the different core files used in the Geane track fitting as shown in Figure 6 and mentioned up above, as well as further detail where necessary.

1. **GeaneReco_module.cc**
   This class is the producer module for the Geane track fitting reconstruction. It pulls in TrackCandidateArtRecords, and produces a TrackArtRecord, TrackStateArtRecords, and a GEANEArtRecord per candidate. During the fitting, the GEANEArtRecord is passed through the various fitting utils files until the track fitting succeeds or fails, updating along the way. The TrackArtRecord and TrackStateArtRecords are then built from the GEANEArtRecord directly. GeaneReco links to GeaneFittingUtils where the track fitting is actually done.
   Methods
   - **produce**
     General overridden produce method. Calls main track fitting code on an event by event basis. Produces data products for downstream use.
   - **trackFitting**
     Short method which first calls down through GeaneFittingUtils into GeaneParamUtils to set up

the initial parameters for fitting, and then calls the method corresponding to the particular fit mode specified in the fcl file. Also has a method call for checking some failure modes after fitting.

2. **GeaneFittingUtils.cc**
   This class contains several methods for fitting tracks with different fit modes. Each such method sets up the specifics of the fit, altering necessary parameters and calling various methods, and then calls the fittingLoop method which iterates and fits the track. mainFit and fullSeqFit do multiple fits, starting with a wire fit.
   Methods
   - **truthLRFit**
     Fits a track to the U and V hits with the left-right sides known from the Monte Carlo simulation. Only works with simulation.
   - **wireFit**
     Fits a track to the wire centers of the hit straws, ignoring left-right information. This is the easiest fit to do on data since it ignores timing information, which affects dca values and errors.
   - **mainFit**
     First does a wire fit to the track in order to gain left-right information, then does a hybrid wire fit with some left-right choices set based on the doublet geometry and predicted parameters from the first wire fit. Finally fits a track to the dca values of the hits with the left-right sides set from the previous fit, updating each iteration.
   - **fullSeqFit**
     First does a wire fit to the track in order to gain approximate track information useful for solving for the left-right sides of the track. Then locks left-right sides based on used fcl parameters, and sets associated errors, leaving the remaining sides as "unknown." Then does a fast check or approximate fit on all U and V unknowns separately in order to determine a set of best possible U and V left-right combinations. Finally takes the best U and V left-right combinations and combines them into full left-right sequences, which are then fitted as normal with the associated dcas and errors. The best final track with the smallest χ^2 is then taken as the fitted track. See the Left-Right section for more detail on this.
   - **fittingLoop**
     The general fitting iteration loop method. Takes in the GEANEArtRecord, the number of passes to fit over, and whether to update the left-right sides each iteration (for mainFit) as parameters. For each loop it will calculate the predicted parameters and propagate the errors of the fit by calling the errorProp method within GeaneParamUtils, then update the left-right sides from the previous iteration if desired, then calculate the angle-corrected measured parameters based on the left-right sides of the track to fit. Finally, with all parameters determined, it will calculate a χ^2 and an improvement for the track by calling the TrackCorrelation method from the class GeaneFitter. The improvement to the track from GeaneFitter serves as the basis for calculating the next set of predicted parameters. Relevant GEANEArtRecord members are filled and then the track is checked to have converged, with fittingLoop iterating if it has not.
   - **checkExtraneousFailureModes**
     A mostly defunct method with the purpose of checking to see whether the tracking has failed in some unseen way. In the past there were some failure modes which have since been removed such as a track reconstructed with overly high momentum. Currently the only failure mode check is to see whether the X positions of the reconstructed track don't match the wire positions, which should not be the case since the former directly comes from the latter. Some coordinate code which may be the root cause of this is commented out due to framework coding constraints (which can probably be gotten around). This method has been left in for the future possibility of checking other potential failure modes.

3. **GeaneFitter.cc**
   This file consists of the main track fitting χ^2 algorithm. It multiplies measured parameters, predicted parameters, error matrices, and transport matrices together to produce a χ^2 for the track and an improvement to the starting paramters. (It constructs the proper full error matrices by adding the

propagated material error matrices and measurement errors to do this.) See the Formalism section for a detailed description of what this class really does. Contained in this class is also the left-right sequence checking code for the fullSeqFit, which does approximate fits with code and math very similar to a normal fit. See the Left-Right section. There is also a fcl parameter matrixDebug which can be used to turn on or off the many large matrix cout debugging statements.

- **TrackCorrelation**
  The main matrix multiplication routines, essentially the math from [1] with improvements. In order to fit the track properly with the correct errors, the matrix multiplication must be done in the XUV space. Since the transport matrices and predicted parameters are generated in the XYZ space as described in the errorProp method of the GeaneParamUtils section, they must be converted using the Jacobian transformation between the two bases. This is described in the Coordinate Systems section, and the variable itself is saved as a member in the GeaneTrackUtils.cc class.

  One other point of note regarding this method is that depending on the matrix multiplications being done, the various loops within the code might either loop over the maximum number of planes (33, a 0 plane plus the 32 measurement planes), or they might loop over the particular planes that were hit. The code should be relatively clear, but if any indices or loop iterators are messed up in any way, then the track fitting will completely fail, sometimes with only very subtle hints as to where exactly the problem lies. (At one point I spent a week hunting down an off-by-one index error believing it to be a memory problem, due to how the Eigen matrices work.) At some point it might be beneficial to standardize this code, but it has been left alone per the reason stated.
- **makeHybridErrMat**
  A method in order to create a hybrid wire / U or V error matrix for the left-right sequence checking. Only called once per track for the U and V sequences.
- **sequenceChecking**
  The main left-right sequence checking method for an individual event. This method is called up to thousands of times as each U or V sequence is checked. See the GeaneLRUtils.cc and Left-Right sections further below for more detail on what this method does.
- **convertToGeVcm, convertToMeVmm**
  Methods for converting Eigen 5 parameter vectors from GeV cm to MeV mm and vice versa.

4. **GeaneParamUtils.cc**
   This class consists of the main methods which modify various parameters for the track fitting. The setupParams method does its namesake, setting up the initial parameters per track before attempting to fit. The errorProp method deals with calculating predicted parameters and associated track matrices within Geant4. Finally, the calcMeasuredParams method deals with setting and correcting the measured parameters of the fit from the incoming dca values and errors of the track hits. Note that this is the only class in the Geane reconstruction framework which links to and uses Geant4. It is in the constructor of this method that the Geant4 world is constructed once per job, the proper magnetic field is associated with the transportation manager, and the error propagation routines are initialized. (The necessary Geant4, art service, and gm2geom header files are included here.)

   Methods

   - **setXPositions, setCoordMap**
     Hacked in methods to grab needed coordinate system information and the X positions of the wire planes (from the art run object) while improving speed of the tracking.
   - **setupParams**
     This method sets up the parameters at the beginning of each track fit. It is here that the TrackCandidateArtRecord and StrawDigitArtRecord information is copied or converted into relevant GEANEArtRecord members. Other members are filled with default or empty values to start. Members include dcas, dca errors, wire positions, hit sides, etc. This method links to the GeaneDummyUtils file, which allows one to acquire the truth information of the track and use for fitting if desired. The starting guess for the position and momentum of the track can be pulled from

the track candidate (calculated using a simple circle fitter to the hit digits) or truth. There are a couple of variables in the GEANEArtRecord which are filled with the same information in this method, as the initial setup is for doing a wire fit to the track. If doing a different kind of fit, the variable changes and method calls in GeaneFittingUtils handles altering the relevant parameters of the fit. There are also a couple of checks in this method to make sure that we're trying to fit a reasonable track, and not one that for instance has more than one hit on a measurement plane.

- **errorProp**
  This method tracks particles along their average trajectories through the region of the tracker detector using the Geant4 error propagation routines with the correct geometry and field. It builds transport matrices, error matrices, and predicted parameters which are the objects used for fitting the track, as described in the Formalism section. It tracks on a plane by plane and step by step basis. These error propagation routines can be used to track particles forwards and backwards, where the forwards tracking is used in the code. (It will be non-trivial to change the code to be able to back-track.) Some extra detail is given in the gm2Geane section.

  To be a little more explicit, it creates Geane free trajectory states which are the objects that are propagated from target plane to target plane, where the target planes are defined to be parallel in X. (As our wire planes are.) Surface trajectory states are then created on those planes, where the orthonormal vectors defining those planes are in the Y and Z directions. These vectors have to be orthornormal in Geant in order for predicted parameters and errors to be calculated correctly. It is later in the fitting code that these track objects are converted to the UV basis that our detector is naturally defined in.

  While the predicted parameters and errors can be grabbed directly from the surface trajectory state on each plane, the transport matrix cannot be. It is on a step by step basis that the transport matrices are generated, with the previous step transport matrix being dropped with the next step. It is for this reason that the transport matrices must be accumulated and multiplied together from each step in order to form the transport matrix describing the change in parameters from one measurement plane to the next. (There is some code in the Geant source code which appears that might do this automatically, but it is commented out and unclear if it works.) These transport matrices are also only defined in the free state coordinate system, and so need to be converted to the surface system that our fitting uses. When creating surface trajectory states on the target planes, the Jacobian transformation matrices from the free to the surface system are generated and can be saved. It is then a simple matter of multiplying these matrices together to get the transport matrices in the surface system. (Once again these are defined in XYZ, but are converted later in the code to be in XUV.) See the Coordinate Systems and Appendix sections for details on these matrix transforms.

  It's also worth mentioning that within this method there are a number of checks to see whether the tracking has failed due to Geant stepping or Geane propagation issues.

- **calcMeasuredParams**
  Method to correct measured parameters from a radial dca value to a U or V value based on the momentum of the track at the hit from the previous fit and approximating a constant field within the straw. Also corrects the dca errors using a simple straight line approximation in a similar vein. See the Appendix section for more details on how this works. Note that this method also fills the measured parameter objects within the fit in the first place, so if this method is not called then the fit won't have any measured hits to fit to.

5. **GeaneDummyUtils.cc**
   This class deals with truth track information, stored in GhostDetectorArtRecords generated by the TrackerDummyPlane service. These GhostDetectorArtRecords store position and momentum in the world coordinate system, and so need to be converted to the Geane coordinate system(s) when comparing the fitted track against truth or inputting any truth parameters into the fit. Called by GeaneParamUtils and some analyzers.

   Methods

   - **fillDummyHits**
     Fills a local variable with the dummy hits from the art data handle, done once so that the utils

file has access to the data products.

- **getTrackDummyHits**
  Method to fill the GEANEArtRecord dummy hits member variable, dummyPlaneHits, with the ordered dummy hits corresponding to the hit measurement planes.
- **fillCS, fillStationStr**
  Passes some needed coordinate system information to the utils file, and the station that the track was in for coordinate transforms to the correct frame.
- **fillDummyHitInfo, fillDigitHitInfo**
  Fills some commonly used variables throughout different methods in the class.
- **fillTruthParams**
  Brute force searches through the GhostDetectorArtRecords to find the hits corresponding to the digits in the track, and sorting them before calling the getTrackDummyHits method. This is kind of an awkward method because the GhostDetectorArtRecord can't store any sort of plane or module identifiers, except for the volume string. These volume names have the form TRACKERDUMMYPLANEPV[a][b][c], with a the scallop number {0, 12, 18}, b the module number {0 - 7}, and c the plane number {0 - 3}. There are also "0" planes in front of each module for recording the "starting" positions of the tracks, defined as being 10 mm (by fcl parameter) in front of the first wire plane of each module. For these 0 planes, c equals 5. It is then a matter of looping through the GhostDetectorArtRecords until the indices and particle ID match the straw digit that was hit and storing those hits into a sorted member variable. There is associated logic to check all of these conditions, and it's not optimized.
- **createStartGuess**
  If desired, this method can be called to create an intial start guess for the momentum and position from the truth information directly. This guess can be smeared variably with the associated fcl parameters if one wishes.
- **fillLRFromTruth**
  This method calculates the true hit sides of the tracks using the GhostDetectorArtRecord positions and the wire information. These true hit sides can then be grabbed by GeaneParamUtils if one wishes to fit to truth, or the final left-right sides can be compared to the truth at the analyzer stage.
- **checkLRAgainstTruth**
  This method checks to see how well the fitting fared when calculating the left-right sides of the track.

6. **GeaneLRUtils.cc**
   This class deals with the left-right aspects of the Geane fitting. It has a couple of methods for functionality regarding hits with small dcas where the error on the hit is large, filling the left-right sides from upstream, the geometry, or the previous fit iteration, and finally a number of methods for different fitting steps within the full sequence checking left-right fit, fullSeqFit. See the fullSeqFit paragraph in the GeaneFittingUtils section for a quick description of how it works, and the Left-Right section for a more detailed description on how fullSeqFit works.

   Methods

   - **lockSmallDCAErrors, lockSmallDCACenters**
     For hits with small dcas below some threshold, sets the fit errors accordingly, and the hit sides to the wire centers. The lockSmallDCACenters method takes as input the inputHitSides GEANEArtRecord member variable which is important to remember when using and modifying this code.
   - **fillLRFromFit**
     Compares the predicted UV parameters to the wire centers, setting the left-right sides to whichever side the predicted track went. It is used in the mainFit fit mode, called each iteration, and helps to increase the number of well fit tracks. This occurs because even if the previous left-right side was incorrect, sometimes the overall fitted track will pick out the correct left-right sides, though this is not always the case.

- **fillLRFromGeom**
  Checks the hit doublets in the track, and if the perpendicular angle of the track (from a previous wire fit) is small or straight enough, then it is known that the track went between the two wires of the doublet, and sets the left-right sides accordingly. See DocDB 6947 for a more detailed description of this.
- **setUnknownSides**
  Default method to set left-right sides to being unknown, the gUnknown value, in preparation for doing a fullSeqFit.
- **fillLRFromNodes**
  Fills the left-right sides from the seed nodes. For this to work the seeds.reconstructPosition fcl variable needs to be set to true, and the left-right information of the nodes needs to be filled upstream somehow. The only fcl paramater that currently does that is the seeds.useGeometryLR fcl variable which sets the sides based on hit doublets, setting the left-right sides as going between the two wires. It does this regardless of angle, and so is incorrect sometimes, unlike the fillLR-FromGeom method above, but has the benefit of not having to do a wire fit beforehand. The seeds.threeDimPositionModel parameter also needs to be set, the default value of "UseDriftCir-cleEstimator" is fine.
- **getUVUnkowns**
  Creates separate vectors of the U and V plane numbers where the left-right sides are unknown, which is useful for the fullSeqFit.
- **getTopSequences**
  Determines the set of best U and V left-right sequences (separately) before full fitting the U and V left-right sequences together. Primarily calls the sequenceChecking method in GeaneFitter. It does this by doing an approximate fit to the $2^{Nu,Nv}$ different combinations of U and V unknown sides, calculating an approximate χ^2, and storing the best 10 sequences. (Takes the 2^N value as an int, turns it into a bitset where the 0 and 1 values stand for left and right respectively, then loops through the bitset trying all combinations until the best are found. At the end this method stores the ints representing the combinations at the end.) This method and the methods it calls within GeaneFitter can take quite a bit a time, and are a good place to look to speed things up. See the Left-Right section for a more detailed description of the math behind the approximate fitter called here.
- **modifyMeasuredParams**
  Called from the method getTopSequences, this sets the measured parameters of the fit to the left or right sides of the wires according the left-right combination being tested in the approximate fitter.
- **getSequenceSides**
  Takes the stored ints representing the best left-right combinations from getTopSequences as input, and sets the sides of the GEANEArtRecord accordingly in preparation for doing the full fit to the combined U and V sequences.

7. **GeaneTrackUtils.cc**
   This utils file contains a number of useful methods for getting track information on wire planes, checking whether a plane is a U or V plane, and has a couple of Eigen variables used in converting from an XYZ frame (with planes staggered parallel in the X direction) to the Geane XUV frame.
8. **GeaneEigenStorageUtils.cc**
   Art data products cannot permanently store eigen objects. This utils file converts Eigen objects to c++ vectors when storing said objects into the art record, and vice versa when reading from the art record. Written by James Mott.

3.6 Analyzers and Filters

There are a number of analyzers and filters which are useful for dealing with the tracks after they have been fitted.

9. **GeanePlots_module.cc**
   The main analyzer/plotting module which runs on the output GEANEArtRecords produced from GeaneReco. This creates many plots including χ^2 distributions, p-value distributions, number of iterations, track parameter histograms, residuals, pulls, etc. There are some fcl paramters to make cuts on different parameters, such as a p-value cut. Specifically there is a fcl variable energyLossCut, which will only plot tracks with a true energy loss of less than the given value, useful for cutting out tracks which lose too much energy (either from actual interaction with material, or the bugged loss in energy even in vacuum - see DocDB 6183 for a quick slide on this unsolved issue.) The default fcl file for this is geanePlotsParams.fcl. See the Plots section for many of the plots that this module produces.

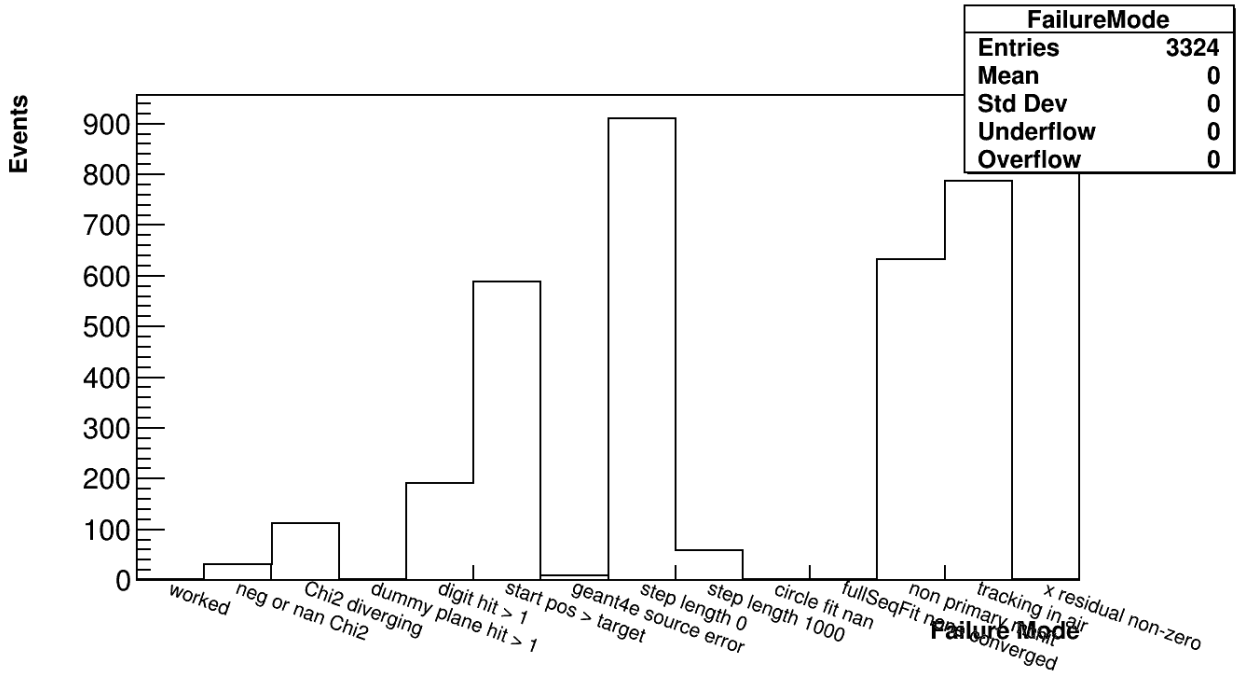
10. **GEANESingleEventViewer_module.cc**
    An analyzer module for looking at the specifics of a fit for single events. Makes parameter plots (measured, predicted, true, X Y Z U V, etc.), error plots, dcas, etc. This is a good analyzer for debugging purposes, to see where the tracking might be having issues. This module is not so good for visualization purposes, though it can do some of that. It will either produces single event plots for all events passed in, or it can produce plots for only a selected set of events using the "eventNum : [#,#,...]" fcl parameter. Note that this option does not currently work with more than one track per event.

11. **GEANEProcessSelection_module.cc**
    This filter module has some functionality for separating out tracks which experienced different physical processes, grabbed from the truth TrajectoryArtRecord data products. Since basically all tracks experience some level of ionization or bremmstrahlung within the straw walls or gas with varying energy deposition, this module doesn't work too great as a filter. Possibly it can be upgraded or improved in some way, but it needs more development before being very useful.

12. **GeaneFailedPlots_module.cc**
    A separate analyzer for looking at events which failed the track fitting. Currently there's only one plot it makes which is a histogram of the various fitting failure modes, gotten from the failureMode member of the GEANEArtRecord. See Figure 7. The different failure modes have changed over time, so there are some empty bins within the histogram as bugs have been fixed or found, or some bin names have been changed. There are some failure modes which can probably be gotten around, but have been left in to try and keep the sample of fitted tracks as clean as possible. A list of the current failure modes is given here, with a number of them being Geant errors:

    - 0 - worked - The track fitting worked. (This bin is not actually filled when making the plot.)
    - 1 - neg or nan Chi2 - The track fitting resulted in a negative or nan χ^2. These are the most important failed tracks since it's not clear exactly what caused these failures.
    - 2 - Chi2 diverging - Triggered when the χ^2 of the current iteration is 10 or more greater than the previous iteration, such that the track fitting is not converging. This is most likely due to a poor initial guess for the track fitting.
    - 3 - dummy plane hit > 1 - A dummy plane was hit more than once by the same particle, leading to an ambiguity in what the true parameters of the track should be. This most likely comes from those events where a track somehow kinks or curves back into the same dummy plane.
    - 4 - digit hit > 1 - The track candidate included more than one hit digit on a single measurement plane, which the track fitting cannot handle. This comes from some problem in the upstream track finding stages.
    - 5 - start pos $>$ target - Geant error - The Geane error propagation start position somehow started after the target (even by a tiny amount), so the tracking will continue indefinitely as it will not reach the target.
    - 6 - geant4e source error - Geant error - The Geane error propagation source code can fail in a number of ways given in G4ErrorFreeTrajState.cc (or the corresponding gm2GeaneFreeTrajState.cc) in the PropagateError method, such as propagation being too close to the Z direction.
    - 7 - step length 0 - Geant error - A step of length 0 was taken. Geant will continue trying to step to the target, sometimes working after a number of trys and sometimes never working.

Figure 7: Shown is a failure modes plot for a random sample of about 200,000 fitted tracks. The number of different failure modes can be seen along the X axis, with varying bin heights. The 0 bin is left unfilled. The "circle fit nan" bin has since been removed, and the "neg or nan Chi2" bin has been reduced. The "x residual non-zero" bin is most likely empty now with code improvements. The sizes and labels of these bins have changed over time with the development of the code and will change going forward into the future.



- 8 - step length 1000 - Geant error - A step of length 1000 was taken, the max step length. This should never occur as steps should be limited by geometry boundaries at the minimum.
- 9 - circle fit nan - Removed, bug since fixed.
- 10 - fullSeqFit none converged - If doing a fullSeqFit, no combinations converged when doing the full fit (either diverging or failing elsewise). Since the fullSeqFit ignores some failure modes (bad combinations might not converge but that's okay since it's a bad combination, and it can be dropped), this bin is needed if all the combinations fail somehow.
- 11 - non primary mchit - Geantish error - Occasionally StrawArtRecord data can come from released photons or ions from the passing particle, and not the actual particle itself, leading to incorrect dcas. Only applicable to truth, and not exactly a bug, but it's nice to be able to remove these poor events.
- 12 - tracking in air - The track fitting is trying to take some steps in air somewhere, which shouldn't happen as the tracker is within the vacuum. Must be coming from some poor initial guess outside the vacuum chamber. These events are just dropped outright, though it might be possible to improve after the first iteration even with some steps in air and salvage some events.
- 13 - x residual non-zero - X residuals for all planes should be exactly equal to 0, but some were seen to be non-zero (though small.) In the interest of cleaning up the sample of fitted tracks they are listed as having failed, but this can probably be removed.

13. **GeanePoorEvent_module.cc**
    A filter to separate out poor tracks, defined as having a pull value (in any of the 5 parameters on plane 0) of greater than 5 (or less than -5). If the track fitting is working correctly, then the pull distribution lies between -5 and 5 (occasionaly a single event can be just outside that range). For more reaslistic data however, particles will lose energy after interaction with material, which the track fitting doesn't

perfectly account for (since it handles averages), and these events can be designated as "poor" though they are real. The more important tracks are the ones which have no energy loss but are still "poor" - these tracks are bugged in some way.

14. **geanePlotsMacro.C**
    A little macro I wrote to combine various plots produced with the GeanePlots analyzer. Useful for looking at things like pull distributions as a function of plane or track length.
15. **haddGEANE.sh**
    This isn't an analyzer, but I figured I'd put a note here on this script which hadds (Root macro) the output files from the GeanePlots analyzer. It is currently deprecated, having been hardcoded to hadd 1000 files (skipping missing files) with delineaters NAME0..999, where the latest updates to the grid and produced file names don't necessarily follow this pattern. However it might be useful to revive this file at some point in the future. For now using the grid or running the analyzers on a list of files to produce one root file at the end is sufficient.

3.7 Other useful fcl files

There is a number of other fcl files which are currently included in gm2tracker which are useful for various analysis or fitting purposes. These are relatively simple and straight forward fcl files which can be deleted, combined, or changed however the user wishes depending on what they are looking at. They are typically used on produced sets of GEANEArtRecords.

1. **RunGeane-midasdata.fcl**
   A simplified fcl file for performing the Geane fitting reconstruction on actual or fake midas data, with fcl parameters set or removed accordingly.
2. **RefitGeane.fcl**
   This file simply refits a previously produced set of track candidates, typically used for refitting failed or bad events from a previous attempt at fitting. One can make changes to the fitting code, refit, and compare to previous results.
3. **makeGEANEPlots.fcl**
   This file will call the main GeanePlots analyzer module for producing plots about the GEANEArtRecords, useful for making plots after tracks have already been produced. (Plots may be added or removed as well.) Note that with the grid updates, it's now necessary to perform the reconstruction and analyzer stages separately, so this fcl file is convenient in that it only includes the latter.
4. **ProcessSelectionGeane.fcl**
   This file will call the filter on different physical processes that events have undergone within the tracker region (ionization, multiple scattering, bremsstrahlung), if one wishes to separate events based on this criteria. (Note that the module that this fcl file calls is not fully finished, nor is it so clear how this filter should work exactly.)
5. **makeGeaneSingleEvent.fcl**
   This file will take produced GEANEArtRecords and call the single event analyzer on the passed in file(s) or a specified set of events. This is useful for analyzing single events to see where rare errors crop up, if tracks are kinking, for debugging purposes, etc. This should not be used for a large number of events.
6. **filterTracks.fcl**
   This file will filter on whether art events successfully produced a GEANEArtRecord or not (succeeded or failed) through the TrackFilter_module.cc, which is useful for reducing the size of art files which contain many events that do not hit the tracker or don't hit enough planes. There is also an option to filter on StrawArtRecords directly before attempting to fit.
7. **fileMerge.fcl**
   This file will simply combine the set of art files that are passed to it, useful for reducing the number of files one has to deal with.

4 Left-Right

This section will describe the fitting modes that can handle determining the left-right sides of the hits within the fitting. The first and shorter section describes the "mainFit" fit mode and the second and longer section describes the "fullSeqFit" fit mode. Some of this information is repeated from the GeaneFittingUtils.cc section describing these fit modes. It is important to remember that while some left-right information for the track may be determined upstream, most of it cannot until a fit of some sort is completed. For us that is the wire fit, which does not require any left-right information in order to work. The wire fit serves as the starting point for determing the left-right sides of the hits since the overall fit will approximately get the left-right sides correct, though not always. Even a single hit side being incorrect can throw the fit off, which is usually readily apparent with resulting high χ^2s or low p-values. Sometimes this incorrect side (or sides) can be determined by looking at the individual plane χ^2s and finding the outliers, though this is not always the case. Due to the smearing of the real dcas from the straw resolution, sometimes it can be very difficult to determine the left-right sides in all hits. There is planned to be a refinement stage in the future, which will be tasked with adjusting the track that is to be fitted, both dealing with left-right sides and removing or adding hits to the fit. Until such a time however, we are unable to fit all tracks correctly with 100% fidelity due to this left-right ambiguity problem.

4.1 mainFit

First a wire fit is completed with a couple of iterations. Then a fit is performed where each iteration takes as its hit sides wherever the previous fit ended up. Something like 2/3rds of the tracks converge nicely with this method, but not all. (In the current iteration of the code there is also an intermediate second wire fit, where the known sides from the doublet geometry and track angle are set, which helps salvage a couple of tracks.) If the left-right sides are only taken from the wire fit a single time, instead of being updated each iteration, then less tracks converge.

4.2 fullSeqFit

This fitting mode takes the longest time to fit a track, because it checks every single left-right combination for all hits in order to find the best track. It starts by first doing a wire fit in order to get approximate track objects (transport matrices, error matrices, predicted parameters) used for making approximate guesses at how good different left-right combinations are. It takes those approximate track objects and a particular combination of left-right choices for the individual hits and constructs a χ^2 for that combination. The set of best χ^2s corresponding to the best left-right combinations are then saved. One can also lock certain hits to left, right, or center depending on how the user is fitting (locking left-right from known geometry, small dcas, etc), and the track fitting will loop over the remaining unknown sides, checking each combination. In order to increase the speed of this code greatly, instead of checking all 2^N combinations directly, the 2^{Nu} and 2^{Nv} combinations are checked separately (leaving the other measured parameter sides as being set to the wires). See Figure 8 for a diagramatic view of exactly how this approximate fitter works for the sequence checking, and read the GeaneLRUtils.cc section for detail on the methods that are used within the fullSeqFit.

Once the set of best U and V sequences have been determined with the approximate fitter, they are combined together and fitted with the full Geane error propagation fitting. For a small sample of fitted tracks, Figure 9 shows where in the set of best 10 saved χ^2s for the different sequences the actual truth sequence is. (For a larger sample of tracks, this plot would fill out more with less empty bins.) It can be seen that the approximate fitter actually guesses the correct U and V sequences a large percentage of the time. When combining the U and V sequences together and full fitting, it is beneficial to scan over some smaller set instead of the full 10 by 10 grid. (Sometimes the approximate fitter can fail to save the correct left-right combination at all if the dca smearing is too large.) The red line in the figure separates out a set of 15 tracks to full fit, which results in high fidelity of fitted tracks. If one wishes an even smaller set of tracks can be full fitted, where more tracks will result in poor fits, but the speed of the tracking improves significantly. See DocDB 6800 for a quick study of track fitting speed vs the level of results, where one should pay attention to the number of tracks within the 0 bin of the p-value plots, representing poorly fit tracks coming from incorrectly chosen left-right sequences.

Figure 8: This figure shows the steps used to determine an approximate set of the best U and V left-right sequences. Note that these steps are followed separately but identically for U and V. The equations here are identical to some of those given in the Formalism section. The sequenceChecking method of the GeaneFitter class is where this is completed in the code. Step 1 is performing a wire fit, done once for all 2^N U or V combinations. The resulting generated transport, error, and covariance matrices along with the predicted parameters from the wire fit are saved for the following steps. Step 2 involves using those aforementioned matrix objects, but now with the measured positions in the calculation replaced with whatever particular left-right sequence is being checked. An approximate fix to the track is then calculated, which can be used in step 3 to determine a new set of approximate predicted parameters. These approximate predicted parameters are then combined with the measured left-right sequence that's being checked in step 4, along with a hybrid error matrix (where if U sequences are being checked, then the V planes have wire errors, and vice versa) to calculate an approximate χ^2 for that particular sequence. All U and V sequences are checked and the best 10, with the lowest χ^2s, of each are saved. The arrows show where matrix objects are used from one step to another. This method while still slow for many combinations, especially high N tracks, is pretty fast because it's simple matrix multiplication over and over again, without actually Geane fitting eaching 2^N combination. (And no inversions.) Note that while not as effective, it is possible to jump from step 1 to step 4, where the predicted parameters remain as those predicted from the wire fit, and calculate an approximate χ^2 for the left-right sequence. This has the benefit of being faster by skipping steps 2 and 3 but fails for more events, and so isn't as useful.

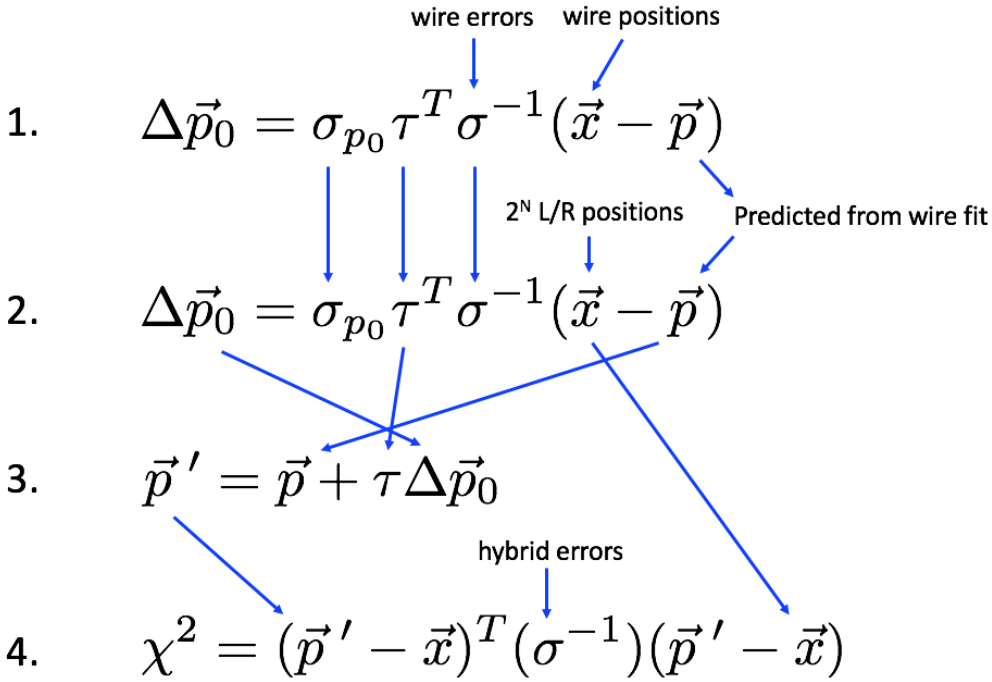
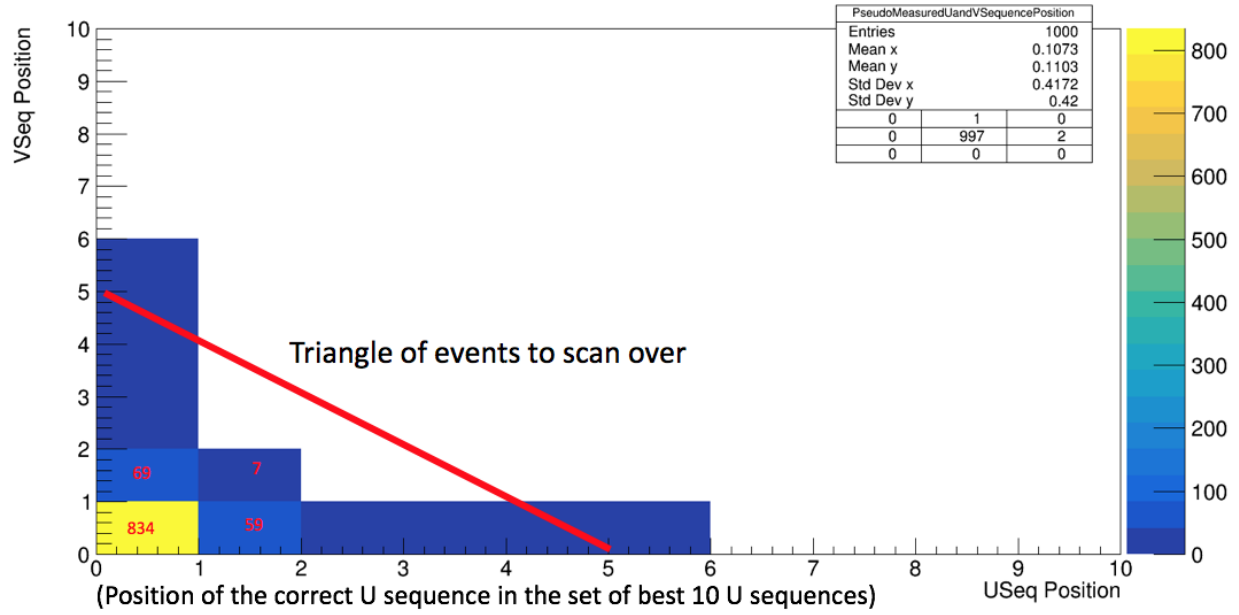


Figure 9: The fullSeqFit mode saves the smallest 10 χ^2s from the approximate fitter for the "best" U and V sequences. For the set of 10 best U sequences, the X bins in this plot represent the location of the true U sequences, and similarly for V. For example, if the 0 bin is filled then the U sequence with the smallest χ^2 is the true sequence, if the 1 bin is filled then the U sequence with the second smallest χ^2 is the true sequence, etc. It can be seen that the approximate fitter does a very good job of actually guessing the true U and V sequences with the bottom 3 bins holding most of the events. Some sequences do end up in the higher bins where the approximate fitter has not done so good of a job at determing the true left-right sequences, including some events which end up in the overflow bins where the approximate fitter has simply failed. The angled red line describes a region of combined sequences to perform the full fitting over in order to try and find the combined true left-right sequence. Note that even then the true sequence may not have the smallest χ^2 in the final fitting due to the smearing of dcas. Note also that hit digits with very small dcas will have very similar χ^2s and fill neighboring bins, though this fact is ignored when locking small dcas to the wire centers.



It is also possible to cut short the sequences that are being full fitted. When there is a couple of very similar sequences, possibly with small dcas, the track fitting results in similar χ^2s. Usually soon after there is a large jump in the full fitted track χ^2 as some left-right choices have been changed, revealing that that particular sequence and the ones after are incorrect. For example the χ^2s might go as 10.2, 11.4, 10.3, 38.9..., and it is at this large jump that one can make a reasonable assumption that the best sequence has been found. This of course does not work perfectly all of the time but it is a good way of eking some more speed out of the tracking. The user can configure the fitting (from base fcl parameters to options such as described here) to optimize vs fit results or speed however desired. This should be combined and studied with the future refinement stage. One might wish to do faster fits to start and refine the poorly fitted tracks before doing more thorough sequence checking and the like.

5 gm2Geane Error Propagation

As with any experiment, the physics of the simulation needs to be attuned to what we observe and the particularities of our detectors. This includes tuning errors and energy loss due to physical processes that particles experience within our trackers. A quick summary of the relevant Geant4 source code and the changes that have been made will be given. Within every default install of Geant4, there is a folder with the

name "error_propagation". As mentioned in the GeaneParamUtils section, this package propagates particles along their average trajectories within a Geant4 world including the magnetic field. Errors and average energy loss due to the presence of material is included, and there are a number of methods to calculate track parameters and errors within different reference frames.

Within the "error_propagation" folder there are a number of classes. Because some changes have been made to this Geant4 source code, several of these classes have been copied into the gm2tracker package and it's those classes that the Geane fitting has been linked to. (A quick aside - the code and physics described in this section are currently contained within the "artg4" package with the folder name "gm2Geane," but will soon be moved into the "gm2tracker" package. I will write this section as if they are already in gm2tracker, but it's good to know this in case the change hasn't been made by the time the reader reads this document, or if the reader wishes to look at old versions of the code.) Because of the linking of those classes to further c++ header classes in the error propagation package, a further number of classes have been copied into this gm2Geane directory with the only changes being header file names. When examining this source code, one will need to look within both the gm2tracker/gm2Geane folder as well as the error_propagation folder in Geant4 in order to get the whole picture. (All these classes of course link to other Geant4 source code dealing with transportation, geometry, etc.)

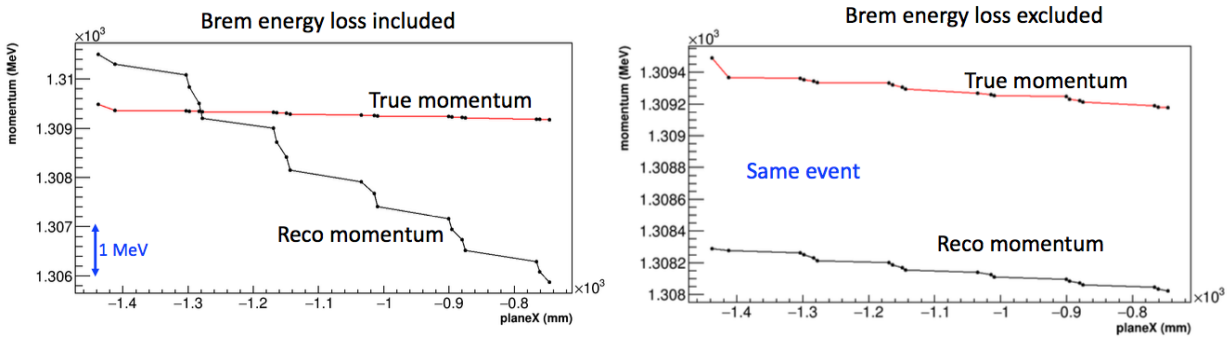
Most of the file names are relatively self explanatory. "gm2GeanePropagator," "gm2GeanePropagatorManager", and "gm2GeaneRunManagerHelper" all deal with the overhead and set up of the error propagation routines and aren't worth going into more detail about. It suffices to say that it's these classes that are instantiated within GeaneParamUtils.cc and where the base methods are called. There are then the classes "gm2GeaneFreeTrajState," "gm2GeaneFreeTrajParam," "gm2GeaneSurfaceTrajState," and "gm2GeaneSurfaceTrajParam" which are derived from some simpler base classes. These deal with the track state and parameter information of the propagated track in the free system and whatever surface system one is working within. They have a number of methods for grabbing these errors, parameters, and matrices. Track errors and other matrices are created within the "gm2GeaneMatrix" and "gm2GeaneSymMatrix" classes, which are 5x5 matrices with units of GeV cm instead of the usual Geant default of MeV mm. It's important to be aware of this in order to convert these units in various parts of the fitting code. There are also a couple of classes related to the physics list, step limiters, and targets that Geant uses when performing the track propagation.

The most important class to understand is probably "gm2GeaneFreeTrajState." The transport matrix is calculated within the long method "PropagateError" with the variable name "theTransfMat", which is then used to propagate the track error from one point to another. It's important to note that this transport matrix is only generated on a step by step basis, and is not saved from target to target. The calculation of this transport matrix (which is just a Jacobian from one curvilinear frame to another) is completed in [4]. In the same file the errors on the track propagation are calculated within the methods "PropagateErrorMSC" and "PropagateErrorIoni" which calculate the error due to multiple scattering and ionization respectively. Note that there is no such method for the error due to bremmstrahlung, which is a rarer process with a larger randomization of energy loss. The Geant4 defaults for these errors are sufficient (but not perfect) for the tracking results so far, but will at some point need to be tuned. (The p-value distribution rises slightly towards 1 indicating an overestimation of errors.) Some basic attempts at error tuning were done using the path taken in Chapter 4 of [2] for the PANDA experiment, which recommends modifying both the multiple scattering error very slightly, and adding an extra term to the ionization errors for delta ray production in thin gaseous detectors. These quick studies with the constants given in that reference didn't result in much changes or improvements and so have been left commented out (but available) in the gm2Geane code. (The delta ray term can be modified to make the error infinitely large if one wishes.) When performing more detailed tuning studies, I highly recommend reading the relevant sections of Chapter 4 of [2] as it contains further derivations and explanations that are not included here.

The class "gm2GeaneSurfaceTrajState" is one of our main hooks into the error propagation code for grabbing the errors and surface trajectory parameters. Those errors and parameters are then transformed to our preferred coordinate system as described in the Coordinate Systems sections. Within this class the "BuildErrorMatrix" method calculates the relevant Jacobian to convert the error from the free trajectory state to the surface trajectory state as described in [4].

Finally, there is the "gm2GeaneEnergyLoss," "gm2GeaneLossForExtrapolator," and "gm2GeaneTablesForExtrapolator" classes which deal with the energy loss of the particle as it is propagated within the Geant4 world. (These files were not originally in the error_propagation directory but in other parts of the Geant4 source code.)

Figure 10: It was noticed during the course of debugging that the Geant4 error_propagation routines were consistently removing too much energy on average from all tracks as they passed through the tracker during reconstruction. It was discovered that the Geant4 tables were taking out too much energy due to bremsstrahlung processes and it was decided to remove this effect. The left plot shows in a red the true particle momentum as a function of X distance through the tracker, and in black the reconstructed momentum before bremsstrahlung was taken out of the reconstruction for a single event. The differing slopes signify the problem. The right plot shows the same event but with bremsstrahlung taken out. Notice the scale change, and that the reconstructed momentum aligns much more readily with the truth.



The full path of function calls will not be included here, it suffices to say that when "gm2GeanePropagator" calls "MakeOneStep" the Geant4 stepping manager is called which then links to the Geant4 transportation and on to the physics calculations. This is done on a step by step basis with the physics list used to calculate the average trajectories of the particles. There is some information on the default energy loss relevant to our track fitting in reference [5] which is useful to understand. However it turned out that the default was including bremmstrahlung energy loss effects which consistently overestimated the energy loss of particles through our thin gaseous detectors. (Thanks to James Mott for confirming this.) In order to remove this effect, within the class "gm2GeaneTablesForExtrapolator" the brem additions to the energy loss were removed in the various methods, including "ComputeElectronDEDX" which is what is called when we propagate positrons. Once this was done, the reconstructed energy loss more closely matches truth, see Figure 10. The simulated energy loss for a distribution of tracks within our detector can be seen in Figure 11, which is what we will expect in reality.

6 Coordinate Systems

There are are number of different coordinate systems to be aware of and understand when dealing with the fitting, both in simulation and reconstruction. For obvious reasons, if the coordinate systems are not properly handled then the fitting will fail. I will summarize here the necessary information for the coordinate systems that are in use. (Other coordinate systems can be integrated and used if one wishes.)

Before fitting, track information comes from the simulation geometry. See Figure 12 for an overview of the simulation tracker geometry and some relevant coordinate system information. Some info from the figure caption is copied here. There is also some relevant information in the RunGeane section. Three trackers live in the Geant4 world with Y vertical, where things like wire center coordinates are recorded as gm2geom::CoordSystem3Vectors which include a reference frame string as well as 3 coordinates. These CoordSystem3Vectors can be transformed from one to another using the transform(css, "world", true) function, where the first parameter is the coordinate system store, the second is the frame to be transformed to, and the third is a boolean value for whether momentum or position is being transformed. (Use "true" for momentum.)

There are 3 specific coordinate systems defined in StrawTrackerCadMesh_service.cc, GeaneTrackerWorld[0,12,18], where these are simple rotations around the vertical Y axis from the three tracker positions to a coodinate system where the tracking planes are parallel in X. These are rotations of -105.52, 74.48, and -15.52 degrees

Figure 11: Shown here is the energy loss between the first and last hit in the tracker from simulation with the full physics list for a distribution of tracks. (Technically momentum magnitude difference.) Note that this is from event generation, and not from reconstruction. Sources of energy loss come from ionization and bremsstrahlung processes, which account for the long Landau tail running off to infinity. The distribution has a mean of approximately 220 keV and a peak centered at about 150 keV, which is reasonable for the material composition of our trackers and tracker gas. More than 50% of the energy loss comes from the mylar walls of the straws as seen in the simulation.

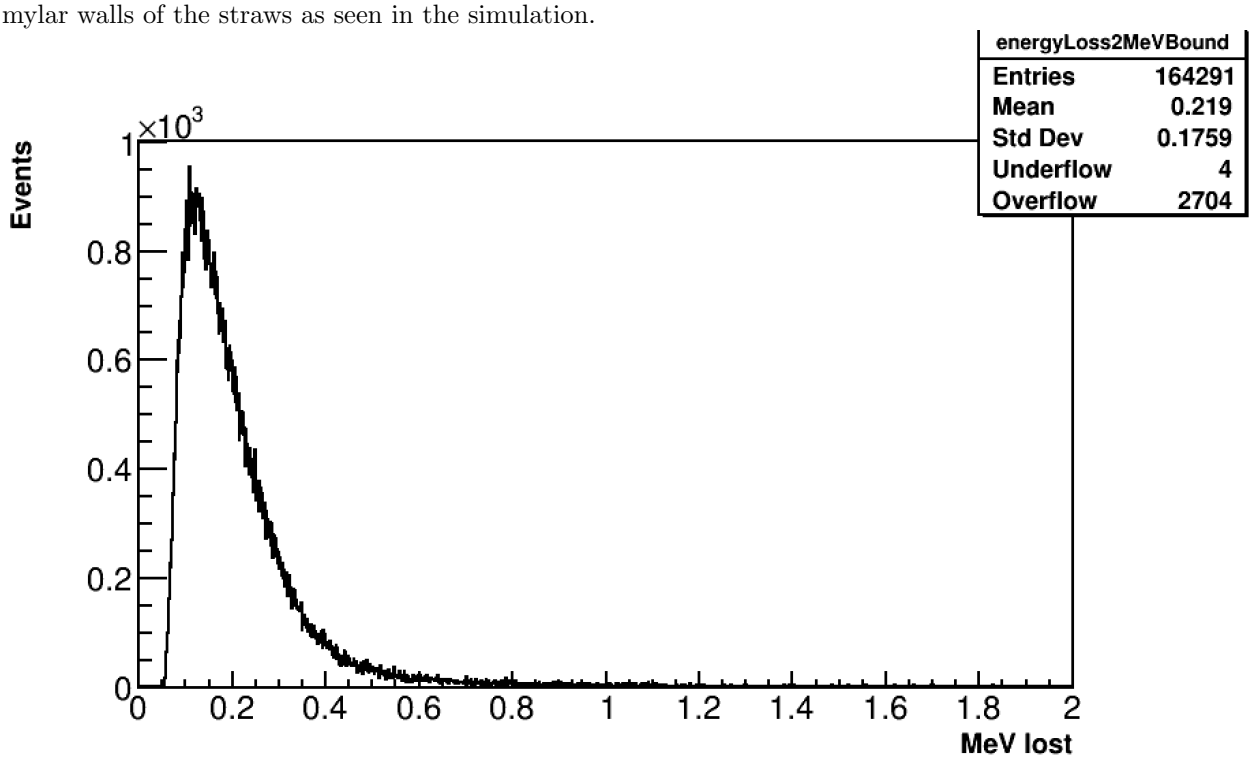
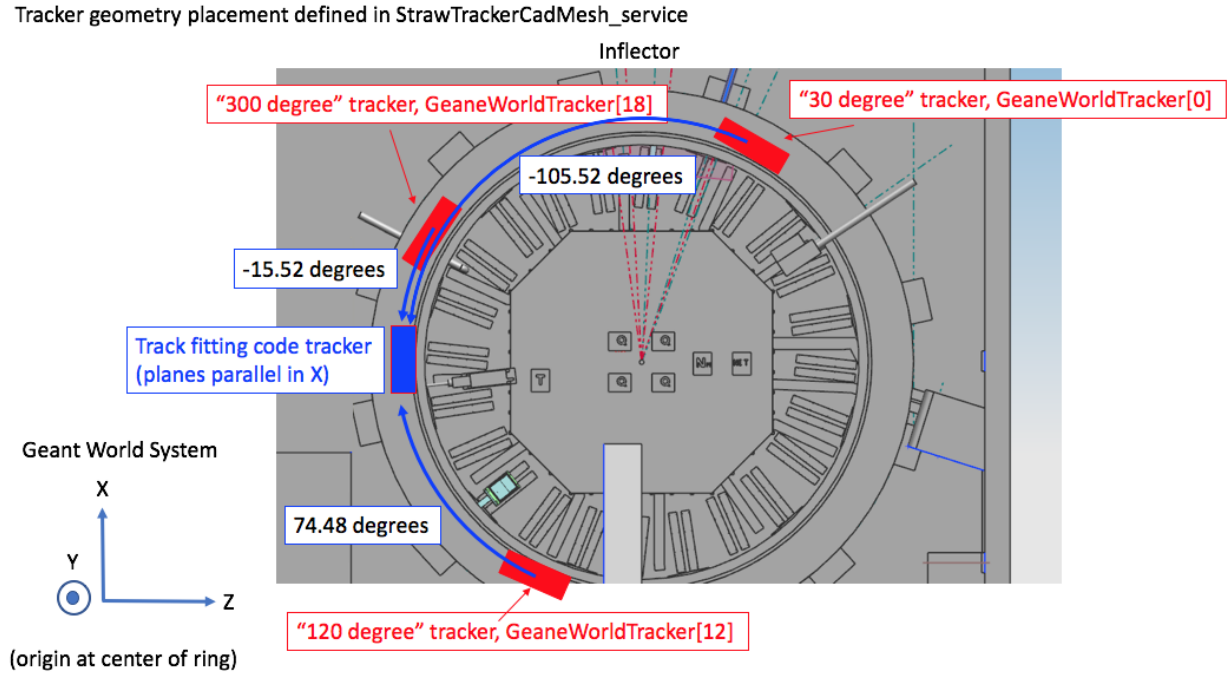


Figure 12: Shown here is a picture of the 3 trackers in the world geometry in their approximate positions. First note the world coordinate system shown in the bottom left of the plot, where the origin lives at the center of the ring. Tracks are generated and read out from the trackers in their three world positions in the red boxes. Due to the reconstruction bug where tracks improperly reconstruct if their momenta is too aligned with the global Z axis, DocDB 4567, the reconstruction rotates the entire arc including tracker 0 to the blue position, where planes are parallel in X, and the fitting is performed in this frame. Track parameters from the 3 separate positions are rotated to this reference frame by the amounts shown in the picture, fitted, and the fitted track at the end is rotated back. (For the parameters to fit, really just the wire center positions are rotated, since the hit dcas are frame independent. The whole Geant geometry needs to be rotated however to include the proper energy loss when using the error propagation routines, so that particles are actually passing through the proper material in the reconstruction.)



respectively, where the numbers come from the geometric placement of the three trackers. (Note that the coordinate system with the tracking planes parallel in X is not actually centered at X = 0.) Only the first, GeaneTrackerWorld[0] is used when performing the error propagation stage of the fitting with the rotateArcTracker fcl parameter of the Arc service. All three are used however when transforming track information to be fitted, and when transforming the fitted track back to the world frame for the extrapolation. Again see Figure 12 for more explanation on this. (There is also another general tracker frame with Z forward which is not in use because of the Z propagation bug.)

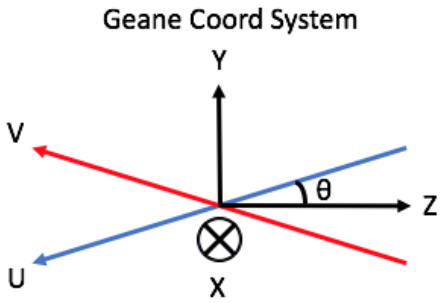
For the fitting, there are a number of track parameterizations involved. I recommend reading [1] which has a quick intro on the different 5 parameter representations of a track, with 1 known parameter. For us the X parameter is known, which is given by virtue of knowing the placement of the trackers and tracking planes. (Neglecting misalignment.) There is also some good information in [2]. I will not describe here the exact meaning of these variables. There is the free system,

$$\frac{1}{p}, \lambda, \phi, y_\perp, z_\perp, \tag{18}$$

and the surface system,

$$\frac{1}{p}, \frac{py}{px}, \frac{pz}{px}, y, z. \tag{19}$$

Figure 13: This picture shows the coordinate system in which the the Geane track reconstruction is performed, in relation to the world coordinate system. The origin remains at the center of the ring, with the tracking planes parallel in X in the reconstruction, going forward in plane number. Y is vertically up, and Z is horizontally to the right. U and V are defined such that they have greater values with higher radii and increasing straw number. These U and V measurement axes are perpendicular to the U and V wire directions.



(In the free system $y_\perp$ and $z_\perp$ are defined with respect to the global Geant4 Z axis as defined in the references.) In the reconstruction, the Geane objects (matrices and parameter vectors) are defined and calculated in the Geant4 error propagation code in the free system. These objects are then transformed internally using Jacobians to the given XYZ surface system, where those Jacobians are defined in [4], which works for us since the tracker planes are parallel and staggered in X. Other surface systems can be given, but we just use XYZ where these are the global X, Y, and Z axes for simplicity. Finally, those objects are transformed to our most natural coordinate system,

$$\frac{1}{p}, \frac{pu}{px}, \frac{pv}{px}, u, v, \tag{20}$$

the XUV system, using the simple Jacobian

$$\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} -\sin\theta & -\cos\theta \\ \sin\theta & -\cos\theta \end{pmatrix} \begin{pmatrix} y \\ z \end{pmatrix}, \tag{21}$$

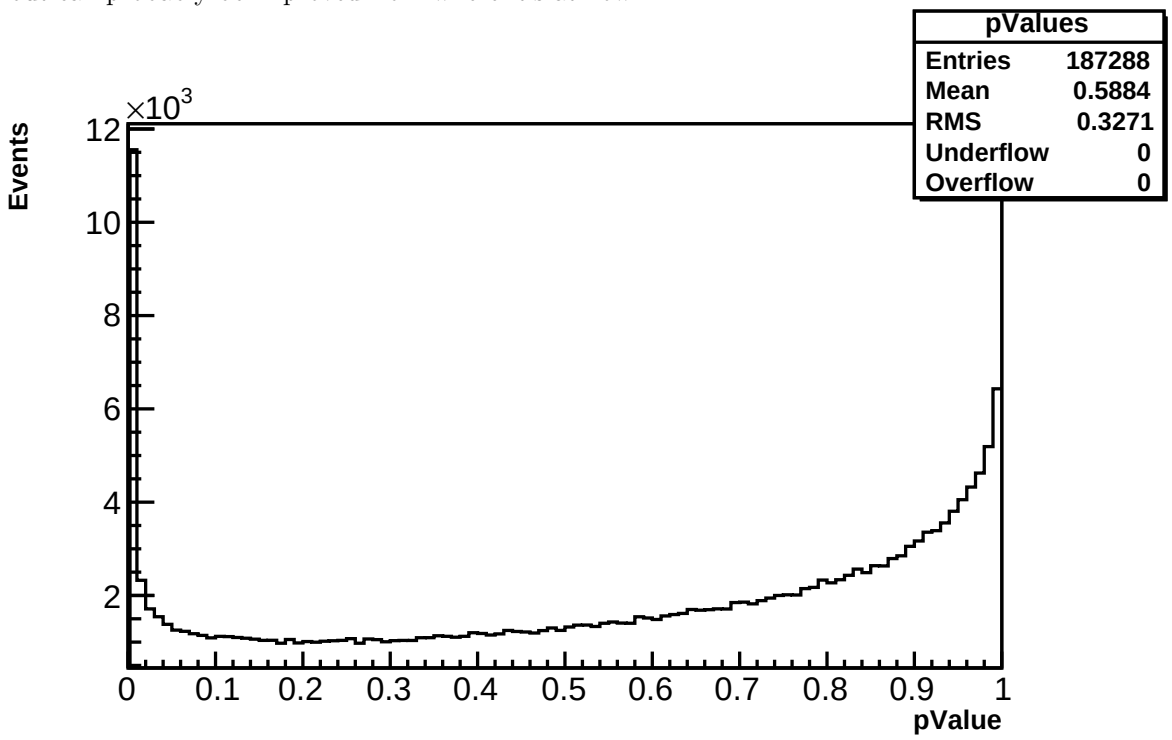
where θ is 7.5°. The 5x5 transformation is just a 1 in the top left corner, and then this matrix in the remaining 2 diagonal blocks. (The reason this XUV system is not given as the initial surface system is because the Geant4 surface trajectory system must be defined in an orthogonal coordinate system.) These two surface systems can be seen in Figure 13. It is in this coordinate system that the fitting is done as per the Formalism section.

Note that the coordinate system variables are switched or renamed among the different references and the Geant4 source code, so one needs to be very careful when trying to make sense of all of this. For instance, we use the XUV system when fitting, but the Geant4 source codes uses the variables UVW for a general surface system, and these variables are not equal.

7 Future To Do

There is always more to do when it comes to tracking. Improvements to the fitting code, optimization, integration of new features up and down stream, and general code development are a few areas that will require work in the future months and years that this code is in use. Listed here are a few specific sections of work that need to be done, though it is not all inclusive. Some of these sections are hinted at in other parts of this document, some of them are included here as possible sections of work (though not necessarily), and some sections are for work that was started but not completed. These are not ordered in any particular way. (These also might have been taken care of by the time the reader looks through this document.)

Figure 14: Shown here is the p-value distribution for fitted tracks using the Garfield parameterization for the dca errors. It can be seen to spike at 0 and rise towards 1, when in an ideal case it would be uniform, indicating that the fitting is not handling the errors exactly correctly. This is probably unavoidable on some level, but can probably be improved from where it's at now.



7.1 Non-Gaussian Garfield Errors

When dealing with real data, the errors on the dca measurements that the straws record are non-Gaussian, stemming from the fact that the drift model is non-linear. To align as best we can with real data, James Mott has developed a Garfield model to deal with these measurement attributes, see DocDB 6171. However, this still causes trouble for our least squares minimizer since it assumes that the fit errors are Gaussian. If one implements the Garfield model directly in the simulation, then the resultant p-value plot rises at both 0 and 1, see Figure 14. For this reason these fit errors and their affect on the track fitting need to be studied, understood, and improved where possible, though it is reasonable that their affects are unavoidable.

7.2 Measurement pulls on all planes

Currently in the track fitting, the measures of how well the track fitting is performing is to look at truth pulls for the parameters on the starting plane as well as the p-value distribution of the fitted tracks. Unit Gaussians for the former and a uniform distribution for the latter prove that the track fitting is working properly. With real data however, the former do not exist, and it is instead more instructive to look at measurement pulls on hit planes. Section 6 of [3] has some information on this, with the mathematical form of these measurement pulls defined as:

$$\frac{\Delta_{meas}}{\sigma} = \frac{p_{fit} - p_{meas}}{\sqrt{\sigma^2_{meas} - \sigma^2_{fit}}}, \tag{22}$$

where all terms are defined on a single plane.

However, if these measurement pulls are implemented as in Equation 22, then the resulting distributions are not unit Gaussians as they should be. If I define them as

$$\frac{\Delta_{meas}}{\sigma} = \frac{p_{fit} - p_{meas}}{\sqrt{\sigma^2_{meas} - \sigma^2_{fit} + \sigma^2_{material}}}, \tag{23}$$

where $\sigma_{material}$ is the error due to the material only, then the resulting distributions look more correct. See DocDB 6261 and DocDB 6306 for calculations and plots regarding this. The problem however is that this doesn't seem to make a whole lot of conceptual sense, adding this extra error term in. Either this extra term is supposed to be added for reasons not fully understood, or it counteracts some mistake in the measurement pull calculation. Understanding and utilizing this will be necessary when it comes to looking at real data.

7.3 Material error tuning

While less serious than dealing with the non-Gaussian measurement errors as described above, it will almost certainly be necessary to deal with the material errors as calculated in the Geant4 error propagation routines, specifically the methods "PropagateErrorMSC" and "PropagateErrorIoni" in the gm2GeaneFreeTrajState.cc file which calculates these material errors. See the gm2Geane section for some more details on this. The reference [2] has some sections on the calculation of these errors for the PANDA experiment, and it is readily observed that these calculations differ slightly from those implemented within Geant. This is supposedly because the default Geant calculations are more attuned to denser scatterers, and not PANDA's straw tube tracker which has some resemblance to our straw detectors. This suggests that we should also be making changes. These modifications are currently programmed into that gm2GeaneFreeTrajState.cc file but commented out. Some quick studies with them turned on were done but no real conclusions were made about whether they should be included in our error calculations or not. It is decently likely that some modifications will need to be included in the error calculations, but that they will not be exactly equal to the ones that PANDA used. It is certainly the case that the p-value distribution right now does indeed rise very slightly towards 1, hinting at not-quite-correct error calculation. The measurement pulls can also be a window into the correctness of these error calculations, but first those pulls need to be understood as described above before they are useful.

7.4 Starting error/covariance

When first creating the free trajectory state for the error propagation, you need to initialize the 5x5 error matrix. For our tracking we've just been using a 0 matrix. That's given us ideal results in a vacuum world, and near ideal results in a material world (with known effects throwing distributions slightly off), indicating that we're performing the track fitting correctly. However it seems like it would make conceptual sense to calculate some error on the track parameters from the initial fitter and use that as an input, as there is an uncertainty in where the track can go that you would want to include. In the same vein, it seems like it would make sense to use the previous fit iteration covariance matrix as the input error matrix for the next iteration. If any of this is included then the fitting results always worsen however, so they have not been included. This work might be unessecary, but I've included this note just in case.

7.5 Initial fitter and effects on Geane fitting

For the track fitting, an initial guess for the starting track parameters is needed before doing the Geane fitting. Currently a simple horizontal circle fit to the wire centers is performed upstream in the track finding code. This circle fitter does a remarkably sufficient job for the vast majority of tracks. There are some tracks which do not converge however, and it is possible that this is because the initial fitter is insufficient. That and the fact that the intial fitter was done simply without much thought means that it might be a good place to eke out some better tracking results. General studies on how good this initial fitter needs to be have yet to be completed also, though it's reasonably likely that it's already sufficient. It is also the case that while no biases due to this have been observed with simulated data, they might rear their heads with real data which is something to keep in mind.

7.6 Bugs and failed tracks

There are always tracks which fail the tracking for one reason or another. See Figure 7. Some of these are due to unavoidable effects like kinking after interacting with material, and some of these are from very fixable sources. Any tracks which fail due to bugs in the tracking code should absolutely be fixed. Known failures included Geant bugs where the step length is 0 or 1000 mm, a 0 transport matrix is returned in the error propagation stage, the error propagation has gotten past the target, or the error propagation returns an internal failure. The exact sources of these bugs are difficult to determine considering that its Geant source code where the problems lie. Some tracks also begin error propagation within the material G4_AIR, and those have been marked to fail. This might be occuring because the intial fitter does so poor of a job that the track starts outside the storage region, or something more sinister might be going on. There is also a bug where in a vacuum world some tracks experience physical processes like ionization or bremmstrahlung (DocDB 6183), which causes some simulated tracks to reconstruct poorly (though not necessarily fail directly). James Stapelton will at some point look more into this since it has some deeper Geant source, but it's good to keep in mind.

There are still some tracks however that return negative or nan χ^2s which have unknown sources. These combined with tracks that diverge leaves a set of tracks that while small are the most important to really understand and fix wherever possible. The latter can probably be improved with a better initial fitter though not fully removed. (There are also some tracks which don't diverge, but also don't converge with a very small $\Delta\chi^2$ between iterations, though these are technically not designated as failures.) Finally there are sources of failures from the upstream tracking code. While most such problems have been fixed, there remains the tracks which end up with more than one digit on a single layer, which the Geane fitting cannot handle. It is also very likely that more unseen issues exist since this is a very complicated chain of track reconstruction.

With further development of the tracking code, both locally, in the upstream reconstruction chain, or even the greater surrounding framework, more bugs or failed tracks will arise. Such new bugs will usually manifest themselves in tracks with negative or nan χ^2s, so if the number of such tracks increases then once can be reasonably sure that a new problem has arisen. This is natural and unavoidable, and it is simply a matter of fixing these issues as they appear. Similarly, as data comes in there is a good chance that new failures will appear, which could possibly be due to more subtle issues. (Though I should mention that no such things were seen with the wire fits on the commissioning data.)

7.7 Physical processes and kinked tracks

As detailed in the Analyzers and Filters section, there is some code that has been partially written regarding the physical processes that the simulated tracks undergo. It would be a good idea to really undertand the effects on the track fitting due to these physical processes. There is a little bit of information on this in DocDB 6099. It is certainly the case that because the Geane fitting creates average trajectory states and adds the error due to material, most tracks are properly treated, but there are some tracks which kink or lose a larger amount of energy and fit poorly. These can usually (but not always) be removed from the tracking results with an energy cut for simulated events. For real data however these effects will still exist. Understanding them and reducing them until they are irreducible will be a necessary step before really looking at all the tracks from data. Certainly for kinked tracks, it should be possible to identify them with some smarter criteria in the analyzer stage other than just looking at event displays as in Figure 15. There is some unfinished code (with some sort of unknown issues) in the GeanePlots module which looks at the TrajectoryArtRecords for this reason, which has been left in for the future.

7.8 Tracker alignment geometry and magnetic fields

Currently in the simulation and reconstruction all trackers have the same perfect placed geometry with wire planes exactly parallel to one another. In reality the different trackers will have slightly different geometry due to imperfect placement into the vacuum chambers as well as slight wire warping. See Figure 16. Once the alignment is completed as best it can be, both physically improved and misalignments recorded (Gleb Lukicov and others), the resultant geometry differences should be coded up and included for the best results possible. It's not clear to me exactly what this will entail, or what complications in the fitting will arise but it should be kept in mind. The errors due to misalignment will somehow need to be folded into the fitting

Figure 15: Shown here is a position plot from the GeaneSingleEventViewer analyzer. The black dots on the red line indicate the true positions of the track that is being reconstructed, with the red line simply connecting the dots. The black line and dots indicates the reconstructed track and positions. The view is side on, with the X axis being the X position of the hits in the reconstruction frame, with particle direction going from left to right as particles pass through the different straw layers. The Y axis indicates the Y position of the track. It can be seen that the true track has upward momentum at the start, hits some material, and kinks at about -1440 mm. (It starts curving back upwards further on because this track was located in a region with some radial field component.) The Geane fitting does it's best to form an average reconstructed track which is seen to be a smooth approximation to the kinked track. This will of course result in a fit with a large χ^2.

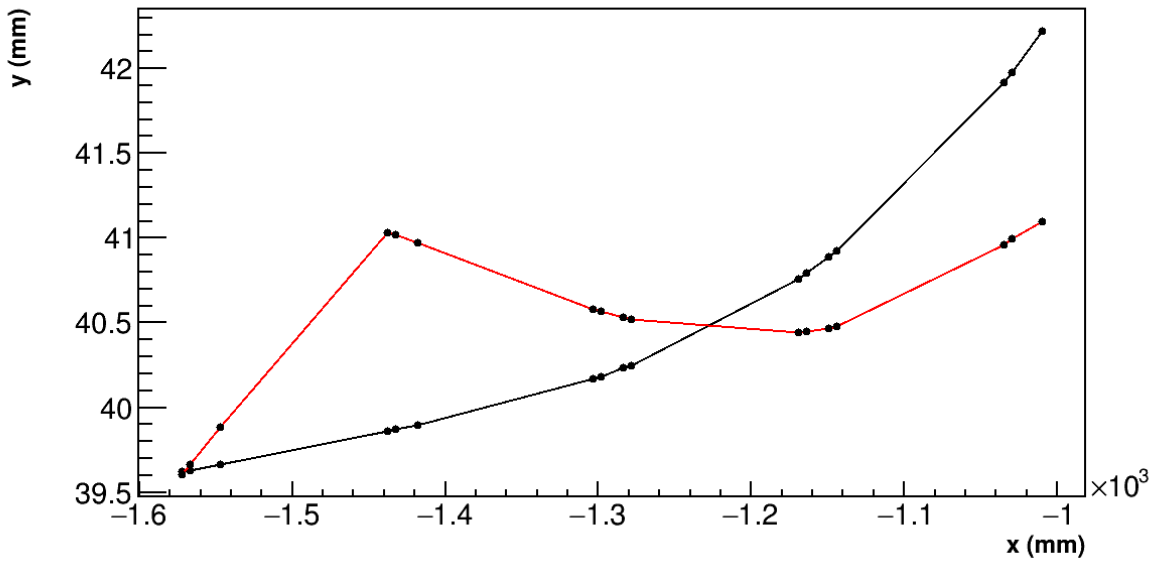
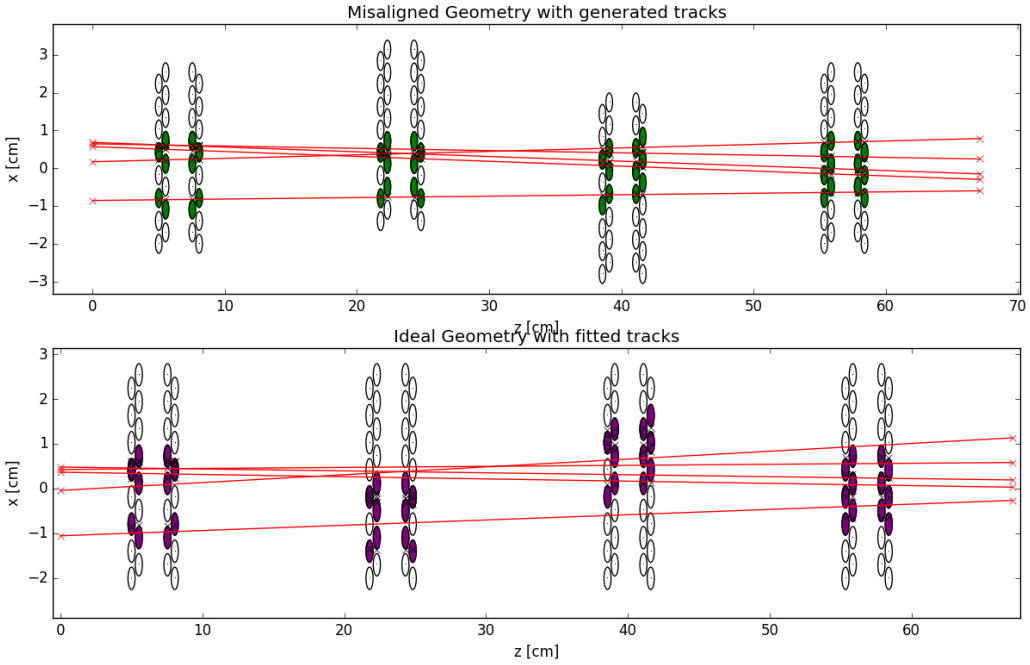


Figure 16: Shown here is a toy model of the alignment code being developed by Gleb Lukicov. The circles represent the straws with the shaded circles being the hits. In the upper picture an exaggerated vertical misalignment in the two center modules can be seen, with the lower picture being the ideal geometry. In red are the straight line fitted tracks to the hit straws. It can be seen that the misalignment in the top picture produces tracks which differ from the true tracks in the lower picture. This means that without the alignment included in the reconstruction we will reconstruct tracks with an excluded error.



errors, either before or after. Certainly the track fitting results without this are a worse representation of reality since the Geane fitting assumes perfect measurement planes, which will not quite be the case.

In a similar vein, the current track fitting incorporates an azimuthally symmetic 2D magnetic field when fitting tracks for the different trackers. This ignores the full 3D field and any differences between the trackers. (Eg. There is no radial field in the simulation currently.) The effects might be negligible, or it might be necessary to include these differences for proper tracking results. It should be on others to calculate, measure, and add these fields to the gm2 simulation, but it will be necessary to update the field setting code in GeaneParamUtils.cc to implement them in the fitting.

7.9 Small dcas

As seen in DocDB 6171 the uncertainty in the dca value measured in a straw when a particle passes close to the wire ($<500\mu m$) is large. For this reason such measurements will be set to the wire center with a larger error. There is code available to do this as described in the GeaneLRUtils section, which has been shown to work. However, exactly what this threshold should be set at, and its combination with the Garfield errors as described above have yet to be implemented. It's important to do this right for correct tracking results, as even a single measurement in a track with incorrectly assigned errors can throw the whole track off.

7.10 5x5 fitter

Deepak Sathyan has created the 5x5 fitter as detailed in [1], as opposed to my more complex full fitter which includes the material correlations. The first results from it are shown in DocDB 6950 though improvements

have been made since then. It can be seen that the results are actually very similar to those from the full fitter. The code itself lives in Deepak's repository on the gm2 machines, and has not been officially added to the feature/trackDevelop branch of the git repository. For speed reasons it will be a good idea to perform the first iterations of the track fitting with this 5x5 fitter which is faster than the full fitter, and only use the full fitter for the laster iteration or so. This should speed up the code significantly without harming physics results.

7.11 Commissioning data

The June 2017 commissioning run for Muon g-2 gathered some very useful data for the trackers. Some of the data and resulting fitted tracks are detailed in DocDB 6992, DocDB 7463, and DocDB 7571. While there is an extent to the usefulness of the data gathered during the commissioning run due to the prevalence of protons, and there is a wealth of work that needs to be done to improve the tracking, it is still very necessary to examine the commissioning data more thoroughly. Doing so could reveal features that the data exhibits that simulation does not, point to bugs or insufficiencies in the tracking code, teach about the tracking in general, etc. Certainly examining this data will be a nice preparation for the data that will come in starting in November and beyond.

7.12 Further left-right work

As detailed in the GeaneLRUtils and Left-Right sections a good amount of work has been done regarding handling the left-right ambiguities of the straw measurements within the track fitting. However this code can certainly be expanded and improved to deal with this better and faster. When updates to the track finding code have been made upstream to help deal with this left-right problem, those updates also need to be integrated with the local Geane fitting code. Further studies beyond DocDB 6800 should also be conducted in order to optimize the tracking results vs speed.

7.13 Refinement

At the time of writing this, the "Refinement" stage of the track reconstruction has not been started. This refinement stage has the job of refining the track by adding hits, removing hits, updating errors, making improvements to left-right guesses, updating T0, and more. While not a direct part of the Geane fitting code, this stage will take information from the Geane fitting results, do its work, and then loop back and refit the track. For this reason, there will be a great deal of work that needs to be done regarding this refinement stage in the future. Be aware that this work will take a very long time (both the updates to the local Geane fitting code and the general refinement stage coding), probably on the order of a year or more, but is a necessary addition to the whole track reconstruction.

7.14 Short track fitter

Tracks which hit a smaller number of planes naturally fit worse with greater uncertainty on the fitted track. There is a hard cutoff in the upstream reconstruction chain at 6 planes hit. There is a noticeable difference in the fit of tracks that hit 7 or 8 planes vs 9 or more. This is because tracks that hit 7 or 8 planes in adjacent modules have a smaller track length which results in larger errors on the momentum measurement. However if those 7 or 8 planes are spread out among modules separated by larger distances then the fit can still be quite good. This is something to be kept in mind when looking at fitting results.

For tracks with 6 or less planes hit however, the hard cutoff has been in place since the development of the Geane fitting code. Because a good number of muons will decay close to the calorimeter and curve sharply through 6 or less planes it will be necessary to be able to fit these "short" tracks. The Geane fitting code should technically be able to handle these short tracks, though the results will certainly be worse, and this needs to be studied. Separately, and maybe with more priority, a specialized short track fitter will probably need to be written.

7.15 Geane extrapolation

After the track fitting stage comes the extrapolation stage, where the fitted track is extrapolated back into the storage region in order to determine the muon beam profile. Saskia Charity has developed the extrapolation code that we've been using. With the goal of having more than one version of every stage of the track reconstruction code however, we'll need a second track extrapolator at some point. The Geant4 Geane error propagation routines provide a nice and straightforward way to do this, which is why I've included a note about it here even though it is not directly related to the Geane fitting. One can pull out and modify the relevant code described in the GeaneParamUtils file to do this. Deepak Sathyan has started this but it requires more work going forward.

7.16 Optimization

While mentioned in the intro to this section, I thought I'd quickly mention optimization. The Geane fitting code like all others needs to be optimized in pretty much all ways. Memory and speed can certainly be improved, maybe even by an order of magnitude or so. I have done very little work regarding this part of the code. Deepak Sathyan quickly and easily improved the track fitting speed by 5x, the first results of which are shown in DocDB 7094. More can and needs to be done.

7.17 Kalman Filter

There is some precedent for using the error propagation matrix objects within a Kalman filter as detailed in the Lavezzi thesis [2], but due to the circumstances following the creation of this code that technique was not followed. At the beginning of development, a (non-Geane) Kalman filter was being started by others, which fell by the wayside a while ago. Since multiple track fitting algorithms will need to be developed for the experiment, this is a good option. There is a lot of useful information in this documentation that would be useful for such an endeavor, though not all the details directly copy over from the least squares global minimization fitter.

7.18 Miscellaneous

I'll mention a couple smaller pieces of work here that shouldn't be forgotten.

In the track fitting, the tungsten wires within the gas should be removed (and just replaced with gas). Due to the uncertainty in the fitted track, it isn't clear whether the track would have gone through a wire or not. If it does in the error propagation however, then a large number is added to the error due to passing through the tungsten material, which is not something we want to include.

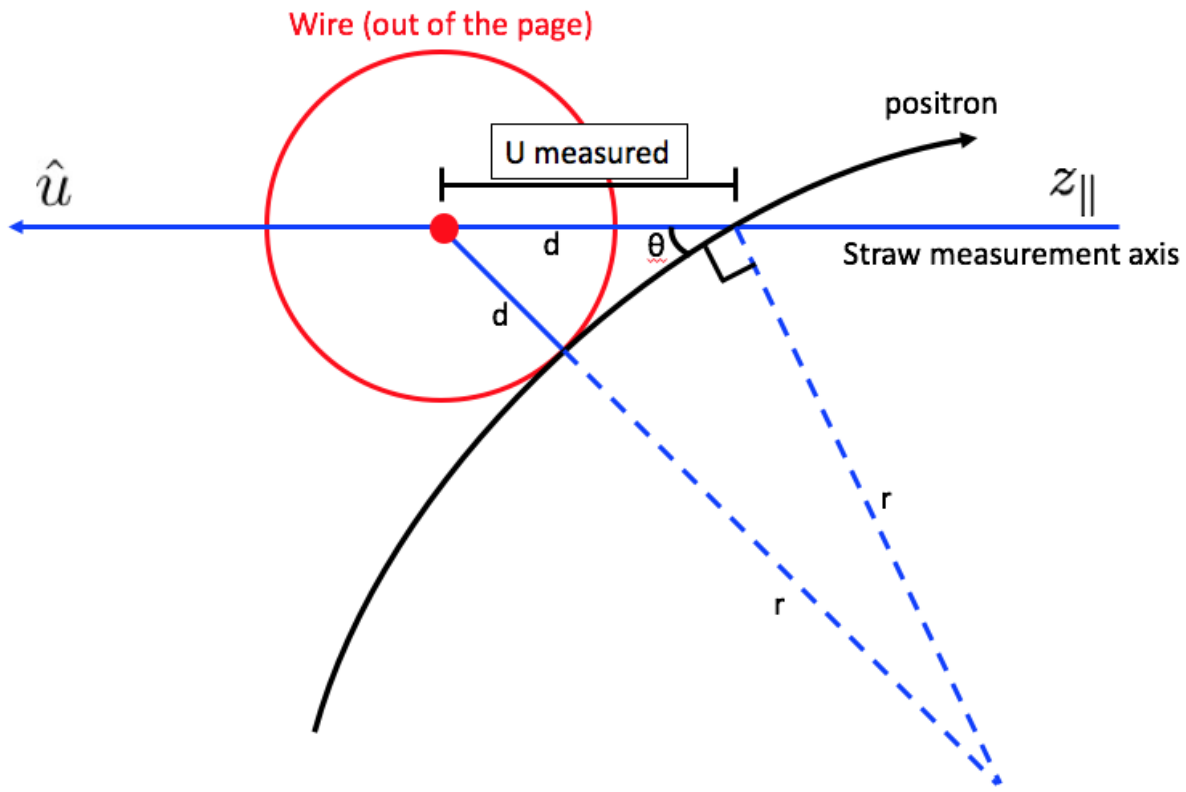
When completing wire fits (very useful for data), there is a discreteness to the results since there is only so many combinations of wires that can be hit. (And a set of hit wires will reconstruct exactly the same track for a wire fit.) This discreteness and its effects should probably be studied in a little more detail, as they seem to introduce some slightly weird effects in the tracking results, as shown in DocDB 7463. Not only that but its possible that this discreteness might be taken advantage of by using some sort of quick look up table to shortcut the wire fit for a known set of wires, per James Mott's suggestion.

A Appendix

A.1 Angular correction

Our straws don't actually measure U and V coordinates directly, but instead measure the distance of closest approach radii deriving from measured hit times. In order to utilize the minimization procedure on measured track parameters these radii must first be converted to U and V parameters, and similarly for the U and V errors. (Note that a future Kalman filter will not be subject to this disadvantage.) This is done in the calcMeasuredParams method at the bottom of the GeaneParamUtils class. These conversion corrections will be dependent on the angle of the track, so it's imporant to note that during each successive iteration, the "measured" parameters are adjusted by the latest "predicted" momenta. It was found that for the error

Figure 17: Shown here is a positron passing through a straw. The desire is to convert the measured parameter d into a U or V position, which can be done by approximating the particle trajectory as a circle in a constant magnetic field over the course of the straw and using trigonometry. Sizes and angles are exaggerated.



correction, a simple straight line correction was sufficient for ideal results. For the position correction, it was found that a constant field correction for curved tracks was sufficient.

To calculate these corrections, first the momentum perpendicular to the straw measurement axis can be ignored since it won't affect the U or V value. Because the positron tracks curve in only one direction through the tracker, one needs to calculate the correction depending on whether the track went to the left or right side of the wire. See Figure 17 for a pictorial representation of the problem. The calculation of the right side correction follows, with the left side correction being calculated in a similar manner.

To solve for the measured U (or V) value, we can use the equation:

$$(r+d)^2 = r^2 + u^2 - 2ru\cos(90+\theta), \tag{24}$$

where the 90 degrees is approximate for large curvature of tracks. The angle θ can be determined from

$$\hat{z_{\|}} \cdot \hat{p_{\|}} = \cos\theta,\ \theta = \cos^{-1}\frac{p_{\|}}{p}, \tag{25}$$

where $p_{\|}$ is the positron momentum parallel to the $z_{\|}$ axis at the wire plane and can be determined within the code. (The $z_{\|}$ axis simply stands for the axis perpendicular to the straw and parallel to the U measurement axis, but opposite in direction. At the end this calculated U value will simply be added or subtracted to the wire center U position depending on which side of the wire the positron travelled.) Using some trig identities and solving for u gives

$$u = -r\sqrt{1-(\frac{p_{\|}}{p})^2} + \sqrt{d^2+2dr+r^2(1-(\frac{p_{\|}}{p})^2)}, \tag{26}$$

for the right side correction, and

$$u = +r\sqrt{1-(\frac{p_\parallel}{p})^2} - \sqrt{d^2+2dr+r^2(1-(\frac{p_\parallel}{p})^2)}, \tag{27}$$

for the left side correction. (Corrections to v are identical.) The radius of the particle circle can be calculated from the circular momentum and magnetic field at the predicted hit position. The straightline correction is done simply using the Pythagorean theorem in a simpler manner, with the correction to the errors then being

$$\sigma'_{uv} = \frac{\sigma_{uv}}{\sqrt{1-(\frac{p_\parallel}{p})^2}}. \tag{28}$$

A.2 Matrix Transformation

Transport matrices are accumulated as

$$T_{mn} = T_{mk}T_{kn}, \tag{29}$$

where the indices can stand for either plane numbers or step numbers, and converted by

$$T_{mn,s} = A_m T_{mn,f} A_n^{-1}, \tag{30}$$

where the indices s and f stand for the surface and free trajectory states respectively, and A is the Jacobian transformation from the free state to the surface state on a particular plane m or n. See [4] for the calculation of these Jacobians. The error matrices in the surface state can be grabbed directly from the Geant code unlike the transport matrices, or if necessary can be converted as

$$\sigma_{m,s} = A_m \sigma_{m,f} A_m^{-1}. \tag{31}$$

A.3 GEANEArtRecord.hh

Table 1: GEANEArtRecord.hh active variables. This is subject to change. GEANEArtRecord contains vectors of variables on planes, as well as larger objects containing information about the whole track. Note that Eigen matrix objects cannot be stored into art data products. For this reason, and for minimal code changes, it was decided to add a data object for each Eigen member, made up of vectors with the word "Data" tagged at the end. The data objects are saved when creating GEANEArtRecords using the GeaneEigenStorageUtils class. In analyzers accessing the GEANEArtRecords, these data objects are then swapped into the Eigen objects using the same utils class.

enum GeaneHitSide *An enum for which side of the wire the track hit is guessed or calculated to have passed. Options are gLeft, gRight, gCenter, gNA_side, and gUnknown. The first three are self explanatory, gNA_side simply means there wasn't a hit, and gUnknown is used within the LR sequence checking part of the fitting code where any hits with said value are looped over to try and find the best choice.*
art::Ptr < gm2strawtracker::TrackCandidateArtRecord > candidate *One track corresponding to one candidate corresponding to one GEANEArtRecord for the whole track.*
std::vector< art::Ptr< gm2truth::GhostDetectorArtRecord > > dummyPlaneHits *Associated dummy plane hits on planes aligned with straw wires, vector consists of hit dummy planes corresponding to hit wire planes (if a straw plane was skipped but that dummy plane was hit, it is not included in this vector. Vector has size N = num hits in straws that form the track +1 for the 0 plane.)*

int failureMode *Different failure modes for failed track reconstruction, 0 means it passed.*
double chi2 *Chi2 for whole track.*
std::vector<double> chi2Planes *Individual chi2s on each plane, vector consists of hit planes with size N. The sum of the chi2s on the planes do not add up to the total chi2, since they don't include the correlations, but get close and can still be used as a measure of goodness of fit at that plane.*
int numIterations *Number of iterations to converge.*
unsigned int dof *DoF of track = number of hit planes - 5 track parameters.*
double chi2DoF *chi2/dof*
double pValue *Fit pValue for whole track.*
std::vector<double> startingTrackParameters *Starting parameters for track: size 6, 3 position then 3 momentum, x y z px py pz, best starting parameters updated after each iteration, starting parameters x position defined before first hit.*
int trackNumPlanesHit *Total number of planes hit.*
int trackFirstPlaneHit
int trackLastPlaneHit
std::vector<int> trackPlanesHitList *List of hit planes, with missed planes excluded from the vector. Ex. 1 2 4 5 8 9*
std::vector<std::vector<double> > wireUVPositions *Wire center U and V postions, first vector is track param vector size 5 (0 1 2 unfilled, 3 is U, 4 is V), second vector is planenumber from 0 - 32 (formatted this way to align with other similar vectors - can probably be reduced.)*
std::vector<double> measuredDCAs *Vector of measured DCAs for hit planes, with size 33. Mainly to hold on to smearing values for now.*
std::vector<double> uncorrErrors *Size 33, errors filled with digit errors or straw diameter errors - uncorrected.*
std::vector<double> UVerrors *Size 33, errors corrected each iteration with angular correction from dca/straw error to UV error - errors that are used in the fit.*
std::vector<gm2strawtracker::GeaneHitSide> inputHitSides *Size 33, LR sides coming from from upstream of the fitting code - and modified within the fitting code depending on the fit mode.*
std::vector<double> dcaErrors *Size 33, dca errors directly copied from straw digit art record.*
std::vector<double> strawDiamErrors *Size 33, vector of straw diameter errors = strawdiameter/$\sqrt{12}$.*
std::vector<double> planeXPositions *Vector of X postions of hit wire planes with size 33 (0 - 32), 0 plane being in front of the first module that was hit.*
std::vector<std::vector<double> > geaneMeasuredParameters *Measured GEANE parameters, first vector is param num 0 - 4, second vector is plane number 0 - 32, units are MeV mm. 1/P, Pu/Pz, Pv/Pz, U, V - only U or V is filled at the start of the GEANE fitting module.*
std::vector<std::vector<double> > geanePredictedParameters

Predicted GEANE parameters, first vector is param num 0 - 4, second vector is plane number 0 - 32, units are MeV mm. 1/P, Py/Px, Pz/Px, Y, Z - all params filled in tracing stage of GEANE fitting module - coord system has to be orthogonal - converted to UV locally in the fitting module.
std::vector<GeaneHitSide> geaneHitSides *Vector of GeaneHitSide enums with size 33 describing the sides of the wires that each hit of the track passed.*
std::vector<Eigen::MatrixXd> geaneTransportMatrices std::vector<std::vector<double> > geaneTransportMatricesData *Units of GeV cm - transport matrices between planes tracked to in GEANE fitting module, 5x5 objects. Vector has size 33.*
std::vector<Eigen::MatrixXd> geaneErrorMatrices std::vector<std::vector<double> > geaneErrorMatricesData *Error matrices on tracked to planes, units GeV cm, size 33.*
Eigen::MatrixXd covarianceTotalInverse std::vector<double> covarianceTotalInverseData *5x5 inverse of total covariance matrix for track. Diagonals represent errors in 5 track paramaters on plane 0.*
std::vector<Eigen::VectorXd> paramPredictedInUVEigen std::vector<std::vector<double> > paramPredictedInUVEigenData *Predicted parameters in UV space as an eigen object (converted from geanePredictedParameters above) for calculation convenience and some LR information storage. Order of vectors is switched here, first is planenum, second is paramnum, units are MeV mm.*
Objects below here are full track objects with larger sizes, held on to for fast sequence checking. Units GeV cm.
std::vector<Eigen::MatrixXd> extendedTransportMatrixBegToEnd std::vector<std::vector<double> > extendedTransportMatrixBegToEndData *Accumulated/combined transport matrices from starting plane to all following planes.*
Eigen::MatrixXd extendedCombinedTransportMatricesTranspose std::vector<double> extendedCombinedTransportMatricesTransposeData *Transpose of larger eigen object composed of above begtoend transport matrices.*
Eigen::MatrixXd extendedReducedMatrix std::vector<double> extendedReducedMatrixData *Total error correlation matrix, reduced to size NxN (N = num planes hit).*
Eigen::MatrixXd extendedReducedMatrixInverse std::vector<double> extendedReducedMatrixInverseData *Inverse of above saved once for LR checking.*
Eigen::MatrixXd extendedModifiedReducedMatrix std::vector<double> extendedModifiedReducedMatrixData *Reduced matrix from above that's going to be modified into a hybrid error matrix separately for U and V fits but that needs to be held onto for all sequences.*

B Plots

It'd probably be good to make one big set of plots to toss into the back for people to look at/reference. All sorts of pulls, parameter plots, residuals, etc. Sort of like Renee's verification pdf.

Also included failed plots and single event viewer plots.

References

[1] E. Nagy V. Innocente M. Maire. *GEANE: Average Tracking and Error Propagation Package*. January 13, 1994.

[2] Lia Lavezzi. "The fit of nuclear tracks in high precision spectroscopy experiments". pp. 57-86. PhD thesis. University of Pavia, Nov. 2007.

[3] E. Nagy V. Innocente. "Trajectory fit in presence of dense materials". In: *Nuclear Instruments and Methods In Physics Research* A.324 (Aug. 1993), pp. 297–306.

[4] W. Wittek A. Strandlie. "Derivation of Jacobians for the propagation of covariance matrices of track parameters in homogeneous magnetic fields". In: *Nuclear Instruments and Methods In Physics Research* A.566 (July 2006), pp. 687–698.

[5] László Urbán Kati Lassila-Perini. "Energy loss in thin layers in GEANT". In: *Nuclear Instruments and Methods In Physics Research* A.362 (Dec. 1994), pp. 416–422.