STAR NOTE

TRACKING FOR THE STAR TPC: DOCUMENTATION AND USER'S GUIDE

Jeffery T. Mitchell and Iwona M. Sakrejda
Nuclear Science Division
Lawrence Berkeley Laboratory

October 5, 1994

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1: INTRODUCTION

The environment created in relativistic heavy ion collisions is quite unique and poses many problems not yet encountered in other collider or fixed target experiments. Two of the most important problems are the high track density and the low average momentum of the produced particles. The latter problem is characterized by multiple scattering and energy loss, which can significantly disturb the topology of the events.

Although the tracking program for the STAR TPC (named tpt) was developed on the basis of a track-following algorithm designed for the ALEPH TPC [1] and modified for the high track density environment for use in the NA36 experiment, it was necessary to design new ways of dealing with the challenges mentioned above.

This document is organized as follows. Chapter 2 provides documentation on the tracking algorithm along with a detailed description of the code. This is not for the faint of heart. The end of Chapter 2 will contain a step-by-step demonstration of what the tracking program does using TAS plots and event displays. Chapter 3 contains an explanation of how to build and run the tracking and evaluation programs along with a description of the parameters and output tables utilized by the tracking code. Chapter 4 continues with a description of the track evaluation philosophy along with documentation on how to build and run a TAS executable that includes the track evaluation module, tte.

If you are just beginning to learn about the STAR TPC tracking, it is recommended that you read Chapter 3 first. If you are interested in learning the basics of the tracking algorithm, it is recommended that you read the introduction to Chapter 2 along with Section 2.3 for a demonstration of the algorithm on real tracks. For those die-hards who want to become experts, Sections 2.1, 2.2, along with all of Chapter 4 is for you.

CHAPTER 2: THE TRACKING ALGORITHM

This chapter provides a detailed documentation of the tracking and evaluation code. It describes all of the algorithms applied and the flow of the program. All of the material is divided into the following sections. Section 1 will describe the algorithms used for the pattern recognition and formation of tracks. Section 2 will describe the algorithms used to fit the reconstructed tracks. Section 3 will track this process step-by-step with event displays for a couple of sample tracks. Section 4 will explain the principles of the track evaluation.

The flow of information between the tracking code (tpt), TPC tracking evaluation code (tte) and other related TAS modules is shown in Figure 2.1. A description of the mct package, which copies ZEBRA hit banks into the appropriate TAS tables can be found in Ref. [2]. A detailed description of the tfs (TPC fast simulator) package and the tphit table can be found in Ref. [3].

A descriptive flowchart of the tracking code is shown in Figure 2.2. This chapter will trace and explain each of the entries in this flowchart in detail. Figure 2.2 serves as a guide for the rest of this chapter.

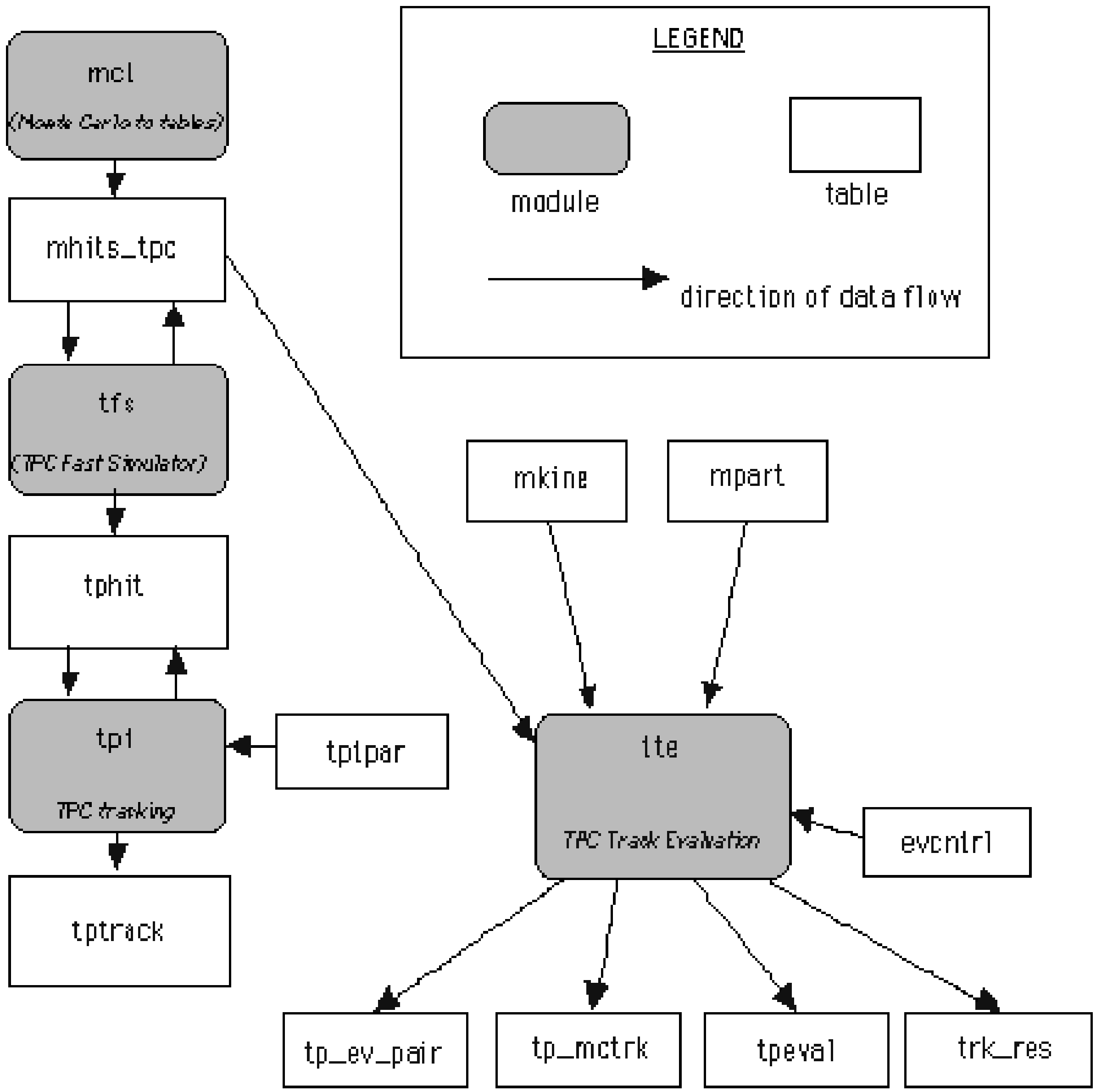

Figure 2.1: Flow of information between the tracking and track evaluation modules.

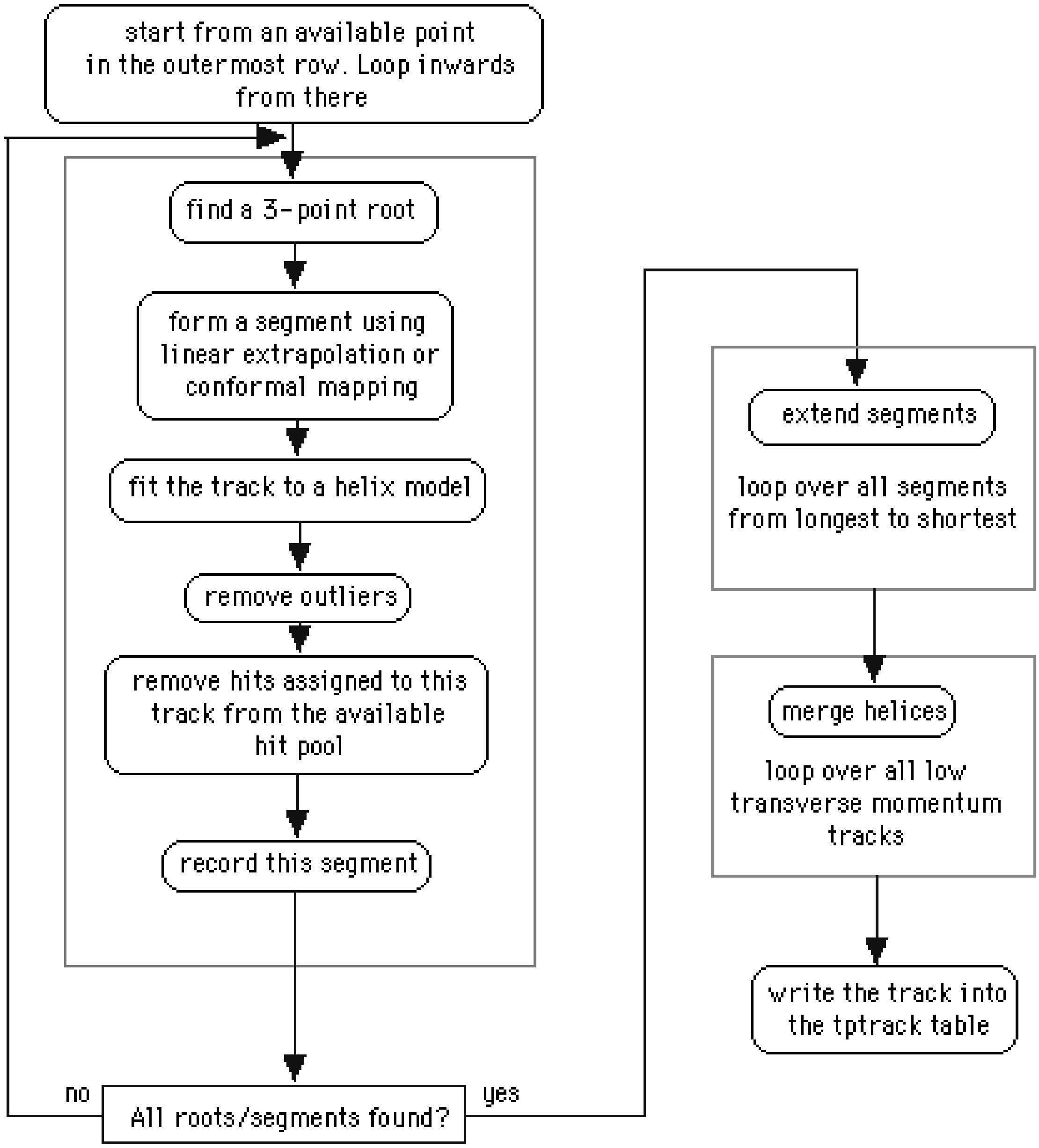

Figure 2.2: Schematic flowchart for the tracking module.

A flowchart of the tpt code can is shown in Figure 2.3. This shows the flow of the program from subroutine-to-subroutine. A brief description of what each subroutine does is outlined below:

TPT_ALL_MAIN: This is the main tracking subroutine. The main program flow as far as real versus ideal (Monte Carlo track fit) tracking is done here.

TPT_DATA_MAIN: This is the main track finding routine for real tracking. All other major subroutines are called from here.

TPT_FORM_SEGMENTS_UV: This performs the segment formation using the conformal mapping method.

TPT_FORM_SEGMENTS: This performs the segment formation using the linear extrapolation method.

TPT_MAKE_SEGMENT_UV: This subroutine will generate one segment starting from a given root using the conformal mapping method to calculate track predictions.

TPT_MAKE_SEGMENT_UV_P: This generates one segment starting from a given root using the conformal mapping method to calculate track predictions with the additional assumption that the track originates from the event vertex.

TPT_PTCRS_UV, TPT_PTCRS_UV_P: This subroutine calculates the crossing of a track with a pad row in the conformal map coordinate system.

TPT_MAKE_SEGMENT: This generates one segment starting from a given root using a linear extrapolation to update the track parameters and assuming that the track originates from the target vertex.

TPT_SECT_FIND: This returns the sector number (1-12) for a given (x,y) coordinate.

TPT_PTCRS: Calculates the crossing point between a row and a track.

TPT_EXTEND_SEGMENTS: Extends the set of segments returned from the track formation phase of the tracking, starting with the longest segment. The helix parameters from the unextended segments are used as track models. Long segments are allowed to steal points from shorter segments, so short segments can be destroyed in this process.

TPT_PROJECT: Adds points to the hit list of the specified track.

TPT_HELIX_PRED: Calculates a coordinate prediction for a track crossing a row using a helix track model.

TPT_CIRCLE_PRED: Calculates a coordinate prediction for a track crossing a row in the bend direction. This is called from tpt_helix_pred.

TPT_MERGE_SEGMENTS: Compares the helix parameters of the fully extended segments and concatanates tracks whose parameters agree within the specified tolerances.

TPT_FIND_ROOT: Forms the three-point track seeds.

TPT_SORT_HIT: Sets up pointers to the tphit table sorted according to the row and sector with which each point belongs. This fills the loc_hit array.

TPT_ROW_SECT_PNT: Sets up information about the z-slices used to reduce the number of points to be sorted through while forming segments.

TPT_HIT_PNT: Addresses hits according to the value of tphit.id. This fills the list_hits array.

TPT_MC_MAIN: This is the main routine for the tracking option of a straight fit of the Monte Carlo tracks.

TPT_FIT_TRACK: This is the main track fitting routine. All other fitting routines are called from here.

TPT_HELIX_FIT: Returns a set of helix parameters for the specified track.

CIRCOV: Calculates the covariance matrix for the circle fit.

LSWLF: This is a least squares straight line fitting routine.

UTCMAT: This is an auxiliary routine used in the circle fit.

UTCFIT: This routine returns a set of circle fit parameters for the specified track.

TPT_FIND_OUTLIER: This subroutine identifies the position of an outlier on a track.

TPT_LOAD_TRACK: Calculates the track parameters in the form necessary to fill the tptrack table.

TPT_LIST_MCHITS: Creates pointers to the Monte Carlo points. This fills the array list_mchits.

TPT_CROSS_FACT: This subroutine evaluates the crossing angle of a track over the specified row and also returns a weight factor for the error calculation.

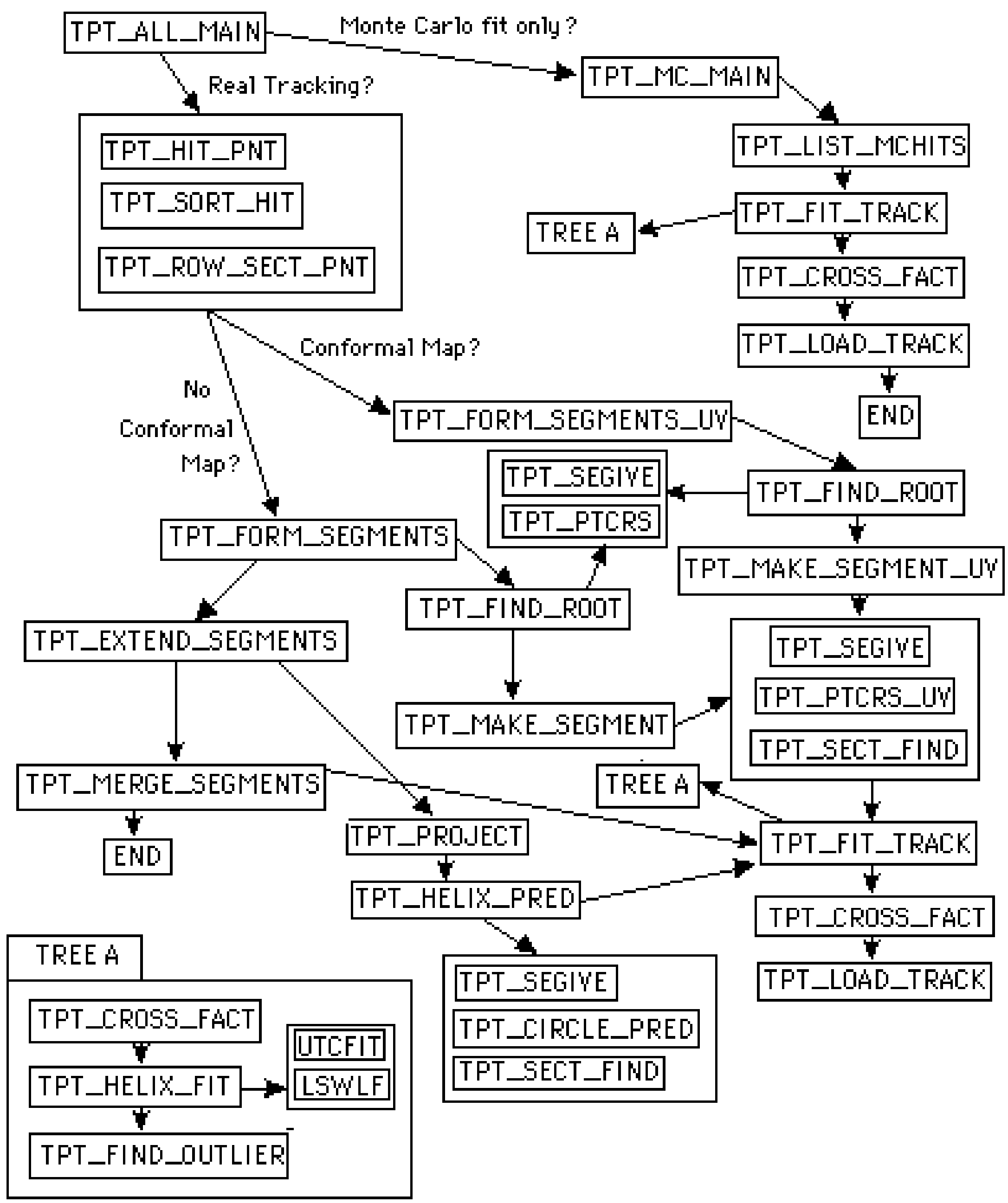

Figure 2.3: Subroutine flow for the tpt module. The function of each subroutine is described in the text.

Section 2.1: Track Finding

Track formation is based on the track following algorithm. The algorithm starts at the outermost row of the TPC where the track density is at its lowest and then proceeds towards the inner rows using the position of the interaction point as a very loose guide. The track finding process contains the following steps:

Root formation: This step generates all possible 3-point *links* (point-to-point connections), or *roots*, starting at a given point. These roots serve as the foundation for the segment formation process. Criteria for accepting links as part of a root are set in the tptpar table. See chapter 3 for more details on the tptpar table. This is described below in Section 2.1.1.

Segment formation: This process tries to form a track candidate starting from a set of links that originate at the same point. Only the best candidate is kept and the points that belong to it are marked as used, that is, they are removed from the available hit pool for segment formation. A helix track model is fitted to the selected points after each segment is formed. This is described below on Section 2.1.2.

Segment extension: This phase makes an attempt to associate more points with the tracks. A helix with parameters fitted in the segment formation step is used as a track model. All the points that are within a specified distance of this model are accepted, added to the segment, and removed from the available point list. This is described below in Section 2.1.3.

Spiral merging: Since low momentum particles lose energy causing their helix parameters to change along their length, a special algorithm must be applied to include these effects and allow for a successful identification of low momentum tracks. This is described in Section 2.1.4.

Section 2.1.1. Root Formation

The first step of the track finding process is that of finding roots from which to build segments. Roots are 3-point links that form the outermost section of the track (to begin with). The projection of these roots into track segments is explained in Section 2.1.2.

Roots are formed from points that lie within the rows specified by the parameters tptpar.irow_root_beg and tptpar.irow_root_end. The algorithm begins looking for hits from which to form roots in the outermost pad row (irow_root_beg) and then search inwards to the innermost pad row (irow_root_end). In this discussion, the three points of a root will be referred to as the outer, middle, and inner points. A root is searched for if the outer point has not previously been attached to a track. Keep in mind that this algorithm will consider the outer point, then form a root, then form a segment, before moving on the the next outer point.

Once an outer point is located, the algorithm searches for points in the adjacent row inwards to the row containing the outer point. In order to reduce the number of points considered in this search, the search is conducted only within slices in z whose width is controlled by the parameter tptpar.nzslic, which is the number of z-slices the width of the TPC is divided into. The width of a z-slice in centimeters is calculated as Δz=2*225.0/tpar.nzslic. The search is conducted within the z-slice containing the outer point along with the z-slices adjacent to it. From this, a list of possible points to be considered as the middle point and inner point of the root is formed. Gaps in the rows of the points of the roots are allowed for by setting the parameters tptpar.g1 and tptpar.g2, which are the maximum allowed row gap size between the outer point and the middle point of the root and between the middle and inner point, respectively.
From the lists of possible middle and inner points, the algorithm applies cuts to determine if valid roots can be formed from them. These cuts are illustrated for the bend plane in Figure 2.4 and for the z-plane in Figure 2.5. In addition, the cuts are derived below.

For this derivation, point 1 is the inner point, point 2 is the middle point, and point 3 is the outer point. Remember that only hits not previously associated with other tracks are considered here. Each middle point is considered individually first. The radii and azimuthal angles of the points are calculated as follows:

$$\Phi_i = \tan^{-1}\left(y_i/x_i\right) \quad (2.1.1) \qquad r_i = \sqrt{x_i^2 + y_i^2} \quad (2.1.2).$$

Differences in radii and rows are defined as $\Delta r_{23}=r_3-r_2$ (2.1.3) and $\Delta row=row_3-row_2$ (2.1.4). The predicted z-position of the middle point of the root is calculated using

$$z_2(pred.) = (z_3 - z_{orig.}) \bullet \frac{r_2}{r_3} + z_{orig.} \quad (2.1.5).$$

Here, $z_{orig}=0$ for tracks originating from the origin of the TPC. The deviation of the predicted middle z value from the z-coordinate of the middle point is

$$\Delta z_2 = \left(z_2 - z_2(pred.)\right) \Big/ \sqrt{1 + \left(\left(z_3 - z_{orig}\right) \big/ r_3\right)^2} \quad (2.1.6).$$

Let F be a scale factor that equals 1 for all rows greater than 14, where it equals 2.5. Then the middle point is subjected to a z-cut whereby the following inequality must hold for it to be accepted at this stage:

$$|\Delta z_2| < F \bullet (tpar.delzlm) \bullet \Delta row \quad (2.1.7).$$

Define the differences $\Delta x_{23}=x_3-x_2$, $\Delta y_{23}=y_3-y_2$, and $\Delta z_{23}=z_3-z_2$ (2.1.8). Then the middle point is subjected to the radial cut as follows:

$$\left(\Delta y_{23}^2 + \Delta x_{23}^2\right) < \left[F \bullet (tpar.delrlm) \bullet \Delta row\right]^2 \quad (2.1.9).$$

If the middle point passes both the z and radial inequalities, then it is used as the basis for finding an inner point of the root.

The algorithm uses the outer and middle points to calculate their straight line intercept [x_1(pred) and y_1(pred)] at the next inner row. This is done using the subroutines tpt_segive and tpt_ptcrs. The value $\Delta row_{12}=row_2-row_1$ (2.1.10) is calculated. The radial tolerance is calculated as

$$D_{row} = F \bullet (tpar.drlm1) \bullet \Delta row_{12} \bullet (tphit.prf_1) \quad (2.1.11).$$

The tolerance in z is $D_z = (tpar.dzlm1) \bullet \Delta row_{12} \bullet F \bullet (tphit.zrf_1)$ (2.1.12).

The inner point is then subjected to the following radial criteria:

$$\left(x_1 - x_1(pred)\right)^2 + \left(y_1 - y_1(pred)\right)^2 < D_{row}^2 \text{ (2.1.13).}$$

If the inner point passes the radial cut, the values $S_{23}=\Delta z_{23}/\Delta r_{23}$ (2.1.14) and

$$z_1(pred) = z_1 - S_{23} \bullet \left(r_2 - \sqrt{x_1(pred)^2 + y_1(pred)^2}\right) \text{ (2.1.15)}$$

are calculated from which the followin cut in z is applied:

$$\left(z_1 - z_1(pred)\right) \Big/ \sqrt{1 + S_{23}^2} < D_z \text{ (2.1.16).}$$

If the inner point passes this final z cut, an entire 3-point root has been successfully formed. The next step is to attempt to form a segment out of this root before continuing to the next root.

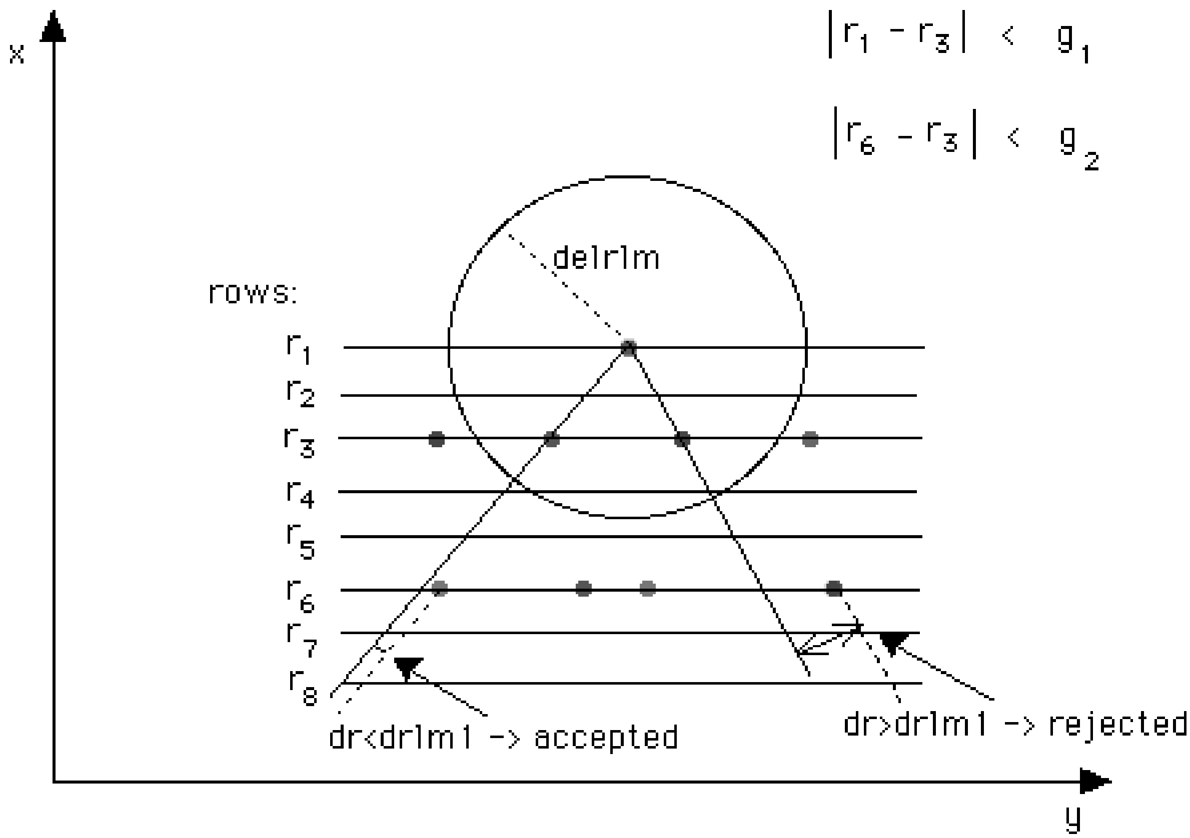

Figure 2.4: Root formation in the bend plane.

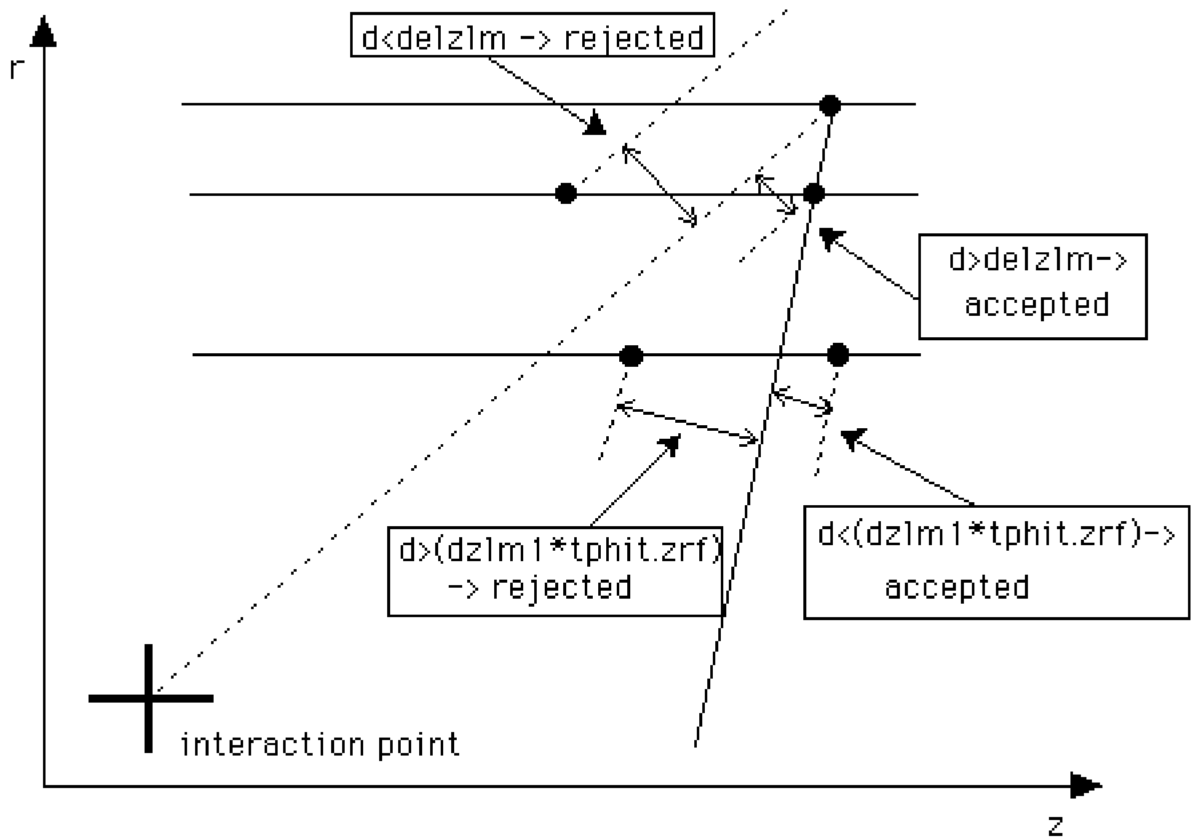

Figure 2.5: Root formation in the r-z plane.

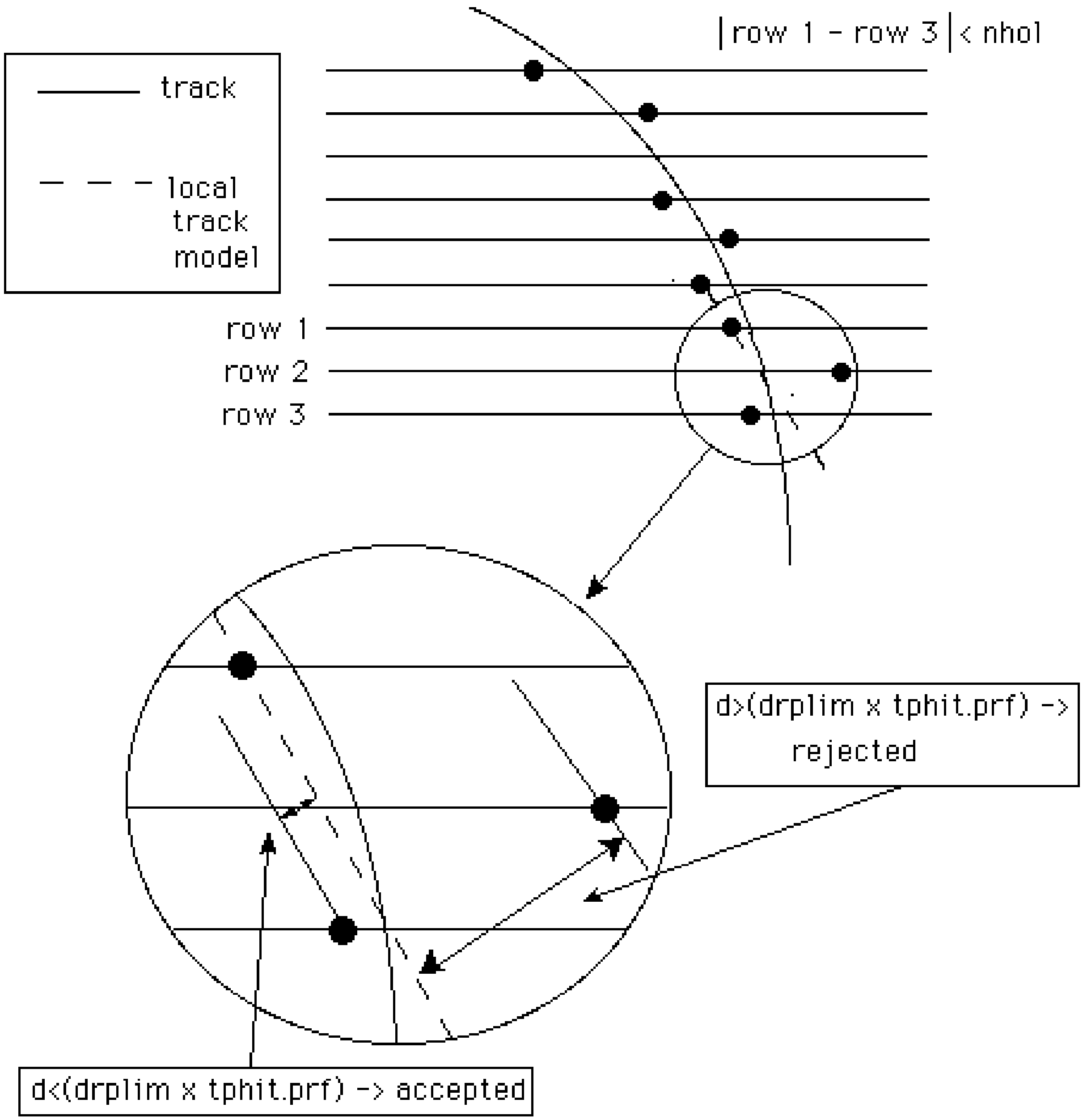

Figure 2.6: Segment formation in the bend plane.

Section 2.1.2. Segment Formation

Once a root has been formed, it is then extended into a track *segment*. There are two methods available for this process. One is using a conformal mapping transformation and can be selected via the parameter tptpar.form_uv. The parameter tptpar.form_uv_p selects segment formation using the conformal mapping with a fixed position of the interaction point. The parameter tptpar.form_xy selects segment formation using a linear extrapolation.

First, the conformal mapping segment formation will be described. Using a simple coordinate transformation [4], a circle of the form

$$(x-a)^2 + (y-b)^2 = R^2 \quad (2.1.17),$$

where a and b are the x and y coordinates of the circle's center and R is its radius, can be transformed into an approximate straight line of the form $v = \frac{1}{2b} - u\frac{a}{b}$ (2.1.18) using the equations

$$u = \frac{(x-x_0)}{\left[(x-x_0)^2 + (y-y_0)^2\right]} \text{ and } u = \frac{(y-y_0)}{\left[(x-x_0)^2 + (y-y_0)^2\right]} \quad (2.1.19),$$

where x_0 is the position of the interaction point.

The algorithm first transforms the three root point coordinates into this conformal map. It then uses the line fitting method described in Section 2.2.2 in order to determine a straight line parametrization of the root. This line fit is then extended to the next inner row, again allowing for possible gaps in rows between points and for sector boundary crossings. From this projection, a prediction for the coordinates of a hit on the next row are determined [x_1(pred) and y_1(pred)]. The subroutines used for determining this prediction are tpt_segive and tpt_ptcrs_uv_p. Again, the z-slice technique described in the previous section is used to form a list of possible points to be included in the segment for the next pad row. Each point is then scrutinized according to the z-coordinate cut:

$$\left|z_i - z_1(pred)\right| < F \bullet (\text{tpar.dzplim}) \bullet (tphit.zrf_i) \quad (2.1.20)$$

where i refers to the point being considered. F has the same values as for the root finding step. The y-coordinate cut is

$$\left[\left(x_i - x_1(pred)\right)^2 + \left(y_i - y_1(pred)\right)^2\right] < \left[F \bullet (\mathrm{tpar.drplim}) \bullet (tphit.prf_i)\right]^2 \quad (2.1.21).$$

If the point passes these two cuts, it is added to this track and removed from the pool of available points for segment formation and the next inner row is considered.

An illustration of the segment formation methodis shown in Figure 2.6. When the conformal mapping method is not selected, the same procedure described above is used, projecting straight lines and forming predictions, but the comparisons are now made in the cartesian coordinate space rather than the mapped coordinate space. Again, only points within the rows tpar.irow_seg_beg and tpar.irow_seg_end are considered.

Section 2.1.3. Extending Segments

Once an initial set of segments have been formed using linear extrapolations, a more rigorous method can be applied to attempt to extend, or add more points, to each segment. This step of the tracking process starts the extension with the longest segment and then considers each segment in turn to the shortest segment. Segments with a number of hits equal to the maximum number of rows being considered will not be extended. If a segment has fewer than tptpar.minfit points associated with it after extension, it is still counted as a track, but tptrack.flag is set to -10 and those hits are retained in the available point pool.

The algorithm (implemented in tpt_project) for segment extension again considers only points within adjacent rows and z-slices to increase processing speed. Only points not currently belonging to a segment are considered. The extension is attempted on both ends of the segment, attempting to extend inwards and then outwards within the TPC. The primary difference in the algorithm for segment extension as opposed to segment formation is that now a helical, rather than a linear, track model is used to obtain projections to the pad row of the hit being considered. This prediction is returned from the subroutine tpt_helix_pred. All available points on the segment are used for the helix projection. Again, two cuts are applied to each point considered. The distance between the prediction and the point is

$$D = \left(x_i - x(pred)\right)^2 + \left(y_i - y(pred)\right)^2 \quad (2.1.22).$$

A cut is applied to this value as $D < \left[(tpar.drstr) \bullet (tphit.prf_i)\right]^2$ (2.1.23).

The second cut utilizes the equations:

$$\alpha = \sin^{-1}\left(0.5 \bullet D/r_0\right) \quad (2.1.24)$$

where r_0 is obtained from the helix parameters of the projection. Also,

$$D_{sp} = 2\alpha \bullet (\tan\lambda) \bullet |r_0| \quad (2.1.25)$$

where λ is the helix dip angle, and

$$D_{sd} = -s_t \bullet (tphit.z_{end} - tphit.z_i) \quad (2.1.26)$$

where end refers to the point at that end of the segment and s_t=-1 if projecting inwards and +1 if projecting outwards. The cut in the z-direction then becomes:

$$\left|D_{sp} - D_{sd}\right| < (tpar.dzstr) \bullet \sqrt{(tphit.zrf)_i^2 + (tphit.zrf)_{end}^2} \quad (2.1.27).$$

A point passing these cuts will be updated to the list of points describing this segment and will also be removed from the pool of available points. The segment extension continues inwards and outwards until either the edge of the TPC is reached or the row gap between points becomes too large.

Section 2.1.4. Helix Merging

After all segments have been extended, an attempt is made to merge segments that have been fragmented (one track divided into two or more segments) during the segment forming process. This is particularly useful for low transverse momentum tracks that have a large path length within the TPC and have a greater chance to become fragmented.

The helix merging algorithm loops through all tracks and orders them according to their radius helix parameter, r_0. Merging helices is accomplished simply by comparing various helix parameter values for each segment. First, the radii are compared according to the cut

$$\left|r_{0,i} - r_{0,j}\right| \Big/ r_{0,j} < 0.1 \quad (2.1.28)$$

where i and j refer to the two segments being compared and the denominator is the largest of the two radii. If two segments pass the radius cuts, the centers of their circle projections (a_{cent} and b_{cent}) are compared next according to

$$\frac{1}{2}\left[\left|a_{cent,i} - a_{cent,j}\right| \Big/ r_{0,(i,j)} + \left|b_{cent,i} - b_{cent,j}\right| \Big/ r_{0,(i,j)}\right] < 0.05 \quad (2.1.29)$$

where again the largest of the two radii is used in the denominator. Finally, the helix parameters are compared according to

$$\left|\tan\lambda_j - \tan\lambda_i\right| < 0.02 \text{ and } \left|\phi_{0,j} - \phi_{0,i}\right| < 0.5 \quad (2.1.30).$$

If all of these criteria are fulfilled, the hits contained within the two segments are assigned to the same segment and a new helix fit is performed using all of the hits. Details on the helix parameters can be found in Section 3.1.2.

Section 2.2: Track Fitting

In a homogeneous magnetic field, tracks are parametrized fairly well by helices. In order to speed up the fitting process, helix parameters are evaluated in two independent procedures. The transverse momentum of a track is evaluated by fitting a circle to a track projection on a plane perpendicular to the magnetic field vector. The momentum along the beam is determined from a straight line fit to the track in s-z coordinates, where s is the track length and z is the track coordinate along the beam.

Section 2.2.1: Circle Fitting

Circles are fitted in the tpt subroutines UTCFIT and UTCMAT. The algorithm used in these subroutines is outlined in Ref [5]. The authors of this reference claim that their fitting procedure is much faster than the regular chi-square minimization while preserving its stability and accuracy. If there are n points with coordinates in the x-y plane defined as (x_i,y_i), where i=1,n. A circle described by the equation $(x-a)^2+(y-b)^2=R^2$ can be drawn by minimizing the mean squared distance of each point from the circle, which is equivalent to minimizing the following functional:

$$L(a,b,R)=\sum_{i=1}^{n}\rho_i^2 \text{ where } \rho_i = \sqrt{(x_i-a)^2+(y_i-b)^2}-R \Big/ \sqrt{w_i} \quad (2.2.1)$$

and w_i is the weight associated with particle i. Since the dependence of ρ_i on the circle parameters is nonlinear, application of the least square method leads to cumbersome computations.

A number of simplified algorithms have been suggested for circle fitting, but they all performed less satisfactorily in terms of the parameter stability and accuracy. One of the most popular methods minimizes the functional:

$$M(a,b,R)=\sum_{i=1}^{n}\left((R+\rho_i)^2-R^2\right)^2=4R^2\sum_{i=1}^{n}\rho_i^2\left(1+\frac{\rho_i}{2R}\right)^2\approx 4R^2L(a,b,R) \quad (2.2.2).$$

One of its drawbacks is that it is more profitable to minimize R^2 than L(a,b,R), which causes the poor stability of this method. Ref. [5] suggest a minimization of the functional:

$$K(a,b,R)=M(a,b,R)\Big/R^2=\sum_{i=1}^{n}\frac{\left(\frac{x_i^2+y_i^2}{R}-\frac{2ax_i}{R}-\frac{2by_i}{R}+\frac{a^2+b^2-R^2}{R}\right)^2}{w_i} \quad (2.2.3).$$

The necessary conditions for this minimization is given by:

$$\frac{\partial K(a,b,R)}{\partial a}=\frac{\partial K(a,b,R)}{\partial b}=\frac{\partial K(a,b,R)}{\partial R}=0 \quad (2.2.4).$$

Evaluation of the partial derivatives of Equation 2.2.3. gives:

$$\sum_{i=1}^{n} \frac{\left(\frac{x_i^2 + y_i^2}{R} - \frac{2ax_i}{R} - \frac{2by_i}{R} + \frac{a^2 + b^2 - R^2}{R} \right)^2}{w_i} \times \left(-\frac{2x_i}{R} + \frac{2a}{R} \right) = 0 ,$$

$$\sum_{i=1}^{n} \frac{\left(\frac{x_i^2 + y_i^2}{R} - \frac{2ax_i}{R} - \frac{2by_i}{R} + \frac{a^2 + b^2 - R^2}{R} \right)^2}{w_i} \times \left(-\frac{2y_i}{R} + \frac{2b}{R} \right) = 0 \text{, and}$$

$$\sum_{i=1}^{n} \frac{\left(\frac{x_i^2 + y_i^2}{R} - \frac{2ax_i}{R} - \frac{2by_i}{R} + \frac{a^2 + b^2 - R^2}{R} \right)^2}{w_i}$$
$$\times \left(-\frac{x_i^2 + y_i^2}{R^2} + \frac{2ax_i}{R^2} + \frac{2by_i}{R^2} + \frac{-2R^2 - a^2 - b^2 + R^2}{R^2} \right) = 0 \quad (2.2.5).$$

If the first two equations of Equation 2.2.5 are divided by 4 and multiplied by R^2, and the third equation is divided by 2 and multiplied by $-R^2$ and $R^2-a^2-b^2$ (also denoted as γ), the following equations are obtained:

$$\sum_{i=1}^{n} \frac{\left(x_i^2 + y_i^2 - 2ax_i - 2by_i - \gamma \right)}{w_i} \times \left(-x_i + a \right) = 0,$$

$$\sum_{i=1}^{n} \frac{\left(x_i^2 + y_i^2 - 2ax_i - 2by_i - \gamma \right)}{w_i} \times \left(-y_i + b \right) = 0,$$

$$\sum_{i=1}^{n} \frac{\left(x_i^2 + y_i^2 - 2ax_i - 2by_i - \gamma \right)}{w_i} \times \left(x_i^2 + y_i^2 - 2ax_i - 2by_i + 2R^2 - \gamma \right) = 0 \quad (2.2.6).$$

Multiplication of the first two equations yields (2.2.7):

$$\sum_{i=1}^{n} \left[-\frac{x_i\left(x_i^2 + y_i^2\right)}{w_i} + \frac{2ax_i^2}{w_i} + \frac{2by_ix_i}{w_i} + \frac{\gamma x_i}{w_i} + \frac{a\left(x_i^2 + y_i^2\right)}{w_i} - \frac{2a^2x_i}{w_i} - \frac{2aby_i}{w_i} - \frac{a\gamma}{w_i} \right] = 0 \text{ and}$$

$$\sum_{i=1}^{n}\left[-\frac{y_i\left(x_i^2+y_i^2\right)}{w_i}+\frac{2by_i^2}{w_i}+\frac{2ax_iy_i}{w_i}+\frac{\gamma y_i}{w_i}+\frac{b\left(x_i^2+y_i^2\right)}{w_i}-\frac{2b^2y_i}{w_i}-\frac{2abx_i}{w_i}-\frac{b\gamma}{w_i}\right]=0.$$

Modifying the last equation by adding and subtracting $2a^2$ and $2b^2$ (2.2.8):

$$\sum_{i=1}^{n}\frac{\left(x_i^2+y_i^2-2ax_i-2by_i-\gamma\right)}{w_i}$$
$$\times\left(x_i^2+y_i^2+2a\left(-x_i+a\right)+2b\left(-y_i+b\right)+2\left(R^2-a^2-b^2\right)-\gamma\right)=0$$

and multiplication gives (2.2.9):

$$\sum_{i=1}^{n}\frac{\left(x_i^2+y_i^2-2ax_i-2by_i-\gamma\right)}{w_i}\times\left(x_i^2+y_i^2+\gamma\right)+$$
$$\sum_{i=1}^{n}\frac{\left(x_i^2+y_i^2-2ax_i-2by_i-\gamma\right)}{w_i}\times\left(-x_i+a\right)+$$
$$\sum_{i=1}^{n}\frac{\left(x_i^2+y_i^2-2ax_i-2by_i-\gamma\right)}{w_i}\times\left(-y_i+b\right)=0$$

The second and third terms in Equation 2.2.9 are equivalent to the first two items of Equation 2.2.6 (i.e., they equal zero) so they can be omitted. Finally, Equation 2.2.9 can be written as (2.2.10):

$$\sum_{i=1}^{n}[\frac{\left(x_i^2+y_i^2\right)^2}{w_i}-\frac{2ax_i\left(x_i^2+y_i^2\right)}{w_i}-\frac{2by_i\left(x_i^2+y_i^2\right)}{w_i}$$
$$-\frac{\gamma\left(x_i^2+y_i^2\right)}{w_i}+\frac{\gamma\left(x_i^2+y_i^2\right)}{w_i}-\frac{2ax_i\gamma}{w_i}-\frac{2by_i\gamma}{w_i}-\frac{\gamma^2}{w_i}]=0$$

Equation 2.2.7 and 2.2.10 can be rewritten using Gauss brackets ($\sum_{i=1}^{n}{x_i^q y_i^p}/{w_i}=\left[x^q y^p\right]$) in the following way:

$$-\left[x\left(x^2+y^2\right)\right]+2a\left[x^2\right]+2b\left[xy\right]+\gamma\left[x\right]+a\left[\left(x^2+y^2\right)\right]-2a^2\left[x\right]-2ab\left[y\right]-a\gamma\left[n\right]=0$$

$$-\left[y\left(x^2+y^2\right)\right]+2b\left[y^2\right]+2a\left[xy\right]+\gamma\left[y\right]+b\left[\left(x^2+y^2\right)\right]-2b^2\left[y\right]-2ab\left[x\right]-b\gamma\left[n\right]=0$$

$$\left[\left(x^2+y^2\right)^2\right]-2a\left[x\left(x^2+y^2\right)\right]-2b\left[y\left(x^2+y^2\right)\right]-2a\gamma\left[x\right]-2b\gamma\left[y\right]-\gamma^2\left[n\right]=0$$

The coordinate system can now be translated in such a way that [x]=0 and [y]=0. This simplifies the preceding set of equations so that they can be written as (2.2.11):

$$-\left[x\left(x^2+y^2\right)\right]+2b\left[xy\right]+a\left[3x^2+y^2\right]-a\gamma\left[n\right]=0$$

$$-\left[y\left(x^2+y^2\right)\right]+2a\left[xy\right]+b\left[x^2+3y^2\right]-b\gamma\left[n\right]=0$$

$$\left[\left(x^2+y^2\right)^2\right]-2a\left[x\left(x^2+y^2\right)\right]-2b\left[y\left(x^2+y^2\right)\right]-\gamma^2\left[n\right]=0$$

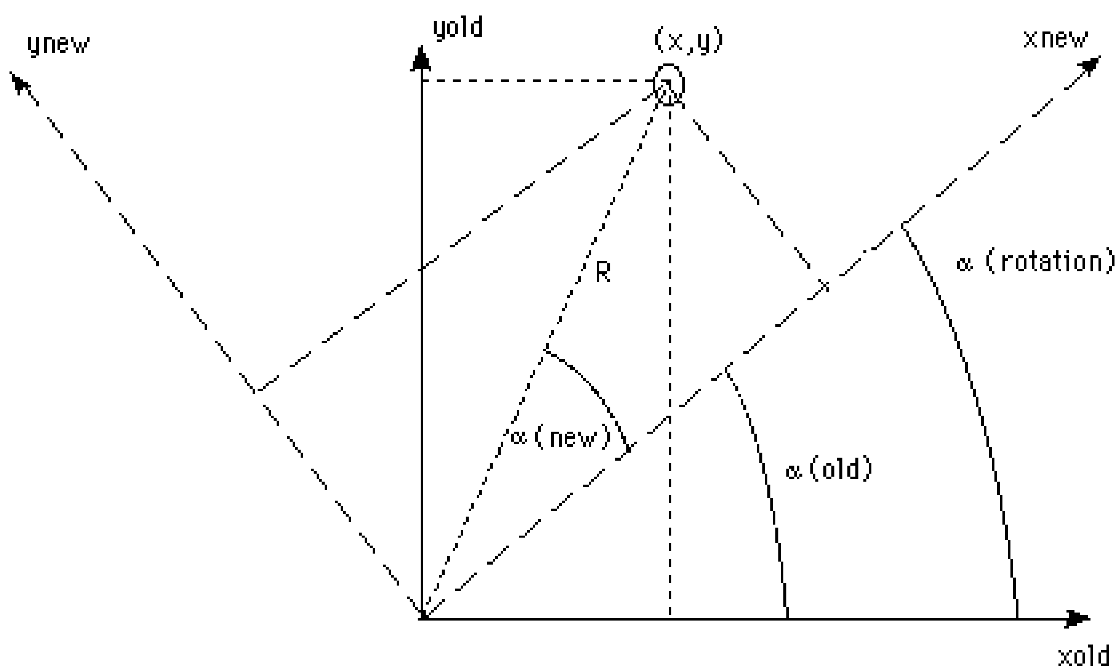

Figure 2.7: Rotation of the coordinate system for circle fitting.

Equation 2.2.11 can be further simplified by rotating the coordinate system in such a way that [xy]=0. The rotation angle, α_{rot}, is defined as $\alpha_{new}=\alpha_{old}-\alpha_{rot}$. The relation between the old and new (rotated) coordinate systems is shown in Figure 2.7. For these coordinate systems, the following equations hold:

$$\begin{cases} x_{new} = r\cos(\alpha_{new}) \\ y_{new} = r\sin(\alpha_{new}) \end{cases}$$

$$\begin{cases} x_{old} = r\cos(\alpha_{old}) \\ y_{old} = r\sin(\alpha_{old}) \end{cases}$$

$$\begin{cases} x_{new} = r\cos(\alpha_{old} - \alpha_{rot}) \\ y_{new} = r\sin(\alpha_{old} - \alpha_{rot}) \end{cases}$$

$$\begin{cases} x_{new} = r\cos(\alpha_{old})\cos(\alpha_{rot}) + r\sin(\alpha_{old})\sin(\alpha_{rot}) \\ y_{new} = r\sin(\alpha_{old})\cos(\alpha_{rot}) - r\cos(\alpha_{old})\sin(\alpha_{rot}) \end{cases}$$

$$\begin{cases} x_{new} = x_{old}\cos(\alpha_{rot}) + y_{old}\sin(\alpha_{rot}) \\ y_{new} = y_{old}\cos(\alpha_{rot}) - y_{old}\sin(\alpha_{rot}) \end{cases}$$

If α_{rot} is denoted by α and $[x_{new}y_{new}]$ is calculated, the following relation holds (2.2.12):

$$\left[x_{new}y_{new}\right] = \sum_{i=1}^{n} \frac{x_i^{new} y_i^{new}}{w_i} = \sum_{i=1}^{n} \frac{\left(x_i\cos(\alpha) + y_i\sin(\alpha)\right)\left(y_i\cos(\alpha) - x_i\sin(\alpha)\right)}{w_i} =$$

$$\cos^2(\alpha)\sum_{i=1}^{n}\frac{x_i y_i}{w_i} + \cos(\alpha)\sin(\alpha)\sum_{i=1}^{n}\frac{y_i^2}{w_i} - \cos(\alpha)\sin(\alpha)\sum_{i=1}^{n}\frac{x_i^2}{w_i} - \sin^2(\alpha)\sum_{i=1}^{n}\frac{x_i y_i}{w_i}$$

If it is required that $[x_{new}y_{new}]=0$, then, from Equation 2.2.12, the rotation angle can be estimated:

$$\cos^2(\alpha)[xy] + \cos(\alpha)\sin(\alpha)\left[y^2 - x^2\right] - \sin^2(\alpha)[xy] = 0$$

which is equivalent to:

$$\left(\cos(\alpha)\sin(\alpha)\left[y^2 - x^2\right]\right)^2 = \left(\sin^2(\alpha) - \cos^2(\alpha)\right)^2\left[xy\right]^2$$

and by replacing $\cos^2(\alpha)$ by $1-\sin^2(\alpha)$, this can be transformed into:

$$\left(1 - \sin^2(\alpha)\right)\sin^2(\alpha)\left[y^2 - x^2\right]^2 = \left(2\sin^2(\alpha) - 1\right)^2\left[xy\right]^2 \quad (2.2.13).$$

Using the denotation $u=\sin^2(\alpha)$, $B=4[xy]^2$, and $A=|[y^2-x^2]|$, Equation 2.2.13 transforms into:

$u(1-u)A^2 - (1-2u)^2\frac{B}{4} = 0$. Multiplying, this becomes:

$$-u^2A^2 + A^2u - \frac{B}{4} - u^2B + uB = 0.$$

Grouping according to powers of u and changing the sign, this becomes:

$$u^2\left(A^2 + B^2\right) + u\left(-A^2 - B\right) + \frac{B}{4} = 0\ ,$$

and the final equation for $\sin^2(\alpha)$ is obtained. To solve this equation, the value Δ is calculated as follows:

$$\Delta = \left(A^2 + B\right)^2 - B\left(A^2 + B\right) = A^2\left(A^2 + B\right)$$

and the roots of this equation can be evaluated:

$u_1 = 0.5\left(1 - \frac{A}{\sqrt{B + A^2}}\right)$ and $u_2 = 0.5\left(1 + \frac{A}{\sqrt{B + A^2}}\right)$.

Since this is a quadratic equation, there are two rotations that fulfill this requirement. As shown in Figure 2.8, both α_1 and α_2 give $[xy]=0$. In order to avoid problems with the charge evaluation, it is important to rotate the track in such a way that either the upper or lower part of the circle is obtained (as opposed to the left or right of the circle), which means that $\left[x_{new}^2\right] > \left[y_{new}^2\right]$ (2.2.14). Let us calculate (2.2.15):

$$\left[x_{new}^2\right]-\left[y_{new}^2\right]=\sum_{i=1}^{n}\frac{\left(x_i^2\right)_{new}}{w_i}-\sum_{i=1}^{n}\frac{\left(y_i^2\right)_{new}}{w_i}=\sum_{i=1}^{n}\frac{\left(x_i\cos\alpha+y_i\sin\alpha\right)^2-\left(y_i\cos\alpha-x_i\sin\alpha\right)^2}{w_i}=$$

$$\sum_{i=1}^{n}\frac{\left(x_i^2-y_i^2\right)\left(1-2\sin^2\alpha\right)+4x_iy_i\sin\alpha\cos\alpha}{w_i}=\left(\cos^2\alpha-\sin^2\alpha\right)\left[x^2-y^2\right]+4\sin\alpha\cos\alpha$$

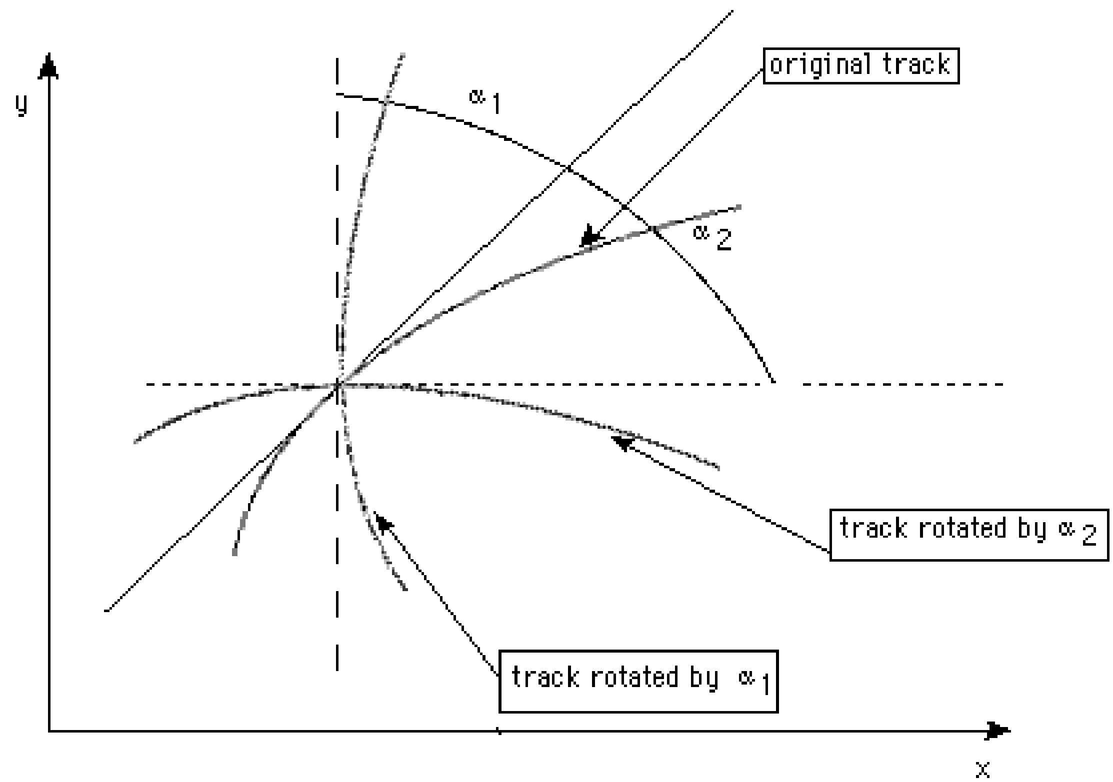

Figure 2.8: Two rotations that satisfy $[x_{new}y_{new}]=0$

Condition 2.2.14 leads to: $\left(\cos^2\alpha-\sin^2\alpha\right)\left[x^2-y^2\right]+4\sin\alpha\cos\alpha>0$. From Equation 2.2.13, sin(α)cos(α) can be calculated as follows:

$$\cos\alpha\sin\alpha=\frac{[xy]}{\left[x^2-y^2\right]}\left(\cos^2\alpha-\sin^2\alpha\right)\ (2.2.16).$$

If, in Equation 2.2.14, sin(α)cos(α) is replaced by a formula calculated from Equation 2.2.16, then Equation 2.2.14 becomes:

$$\frac{\cos^2\alpha - \sin^2\alpha}{\left[x^2 - y^2\right]}\left(\left[x^2 - y^2\right]^2 + 4\left[xy\right]^2\right) > 0.$$

Since $[x^2-y^2]^2+4[xy]^2$ is always positive, then the following must hold:

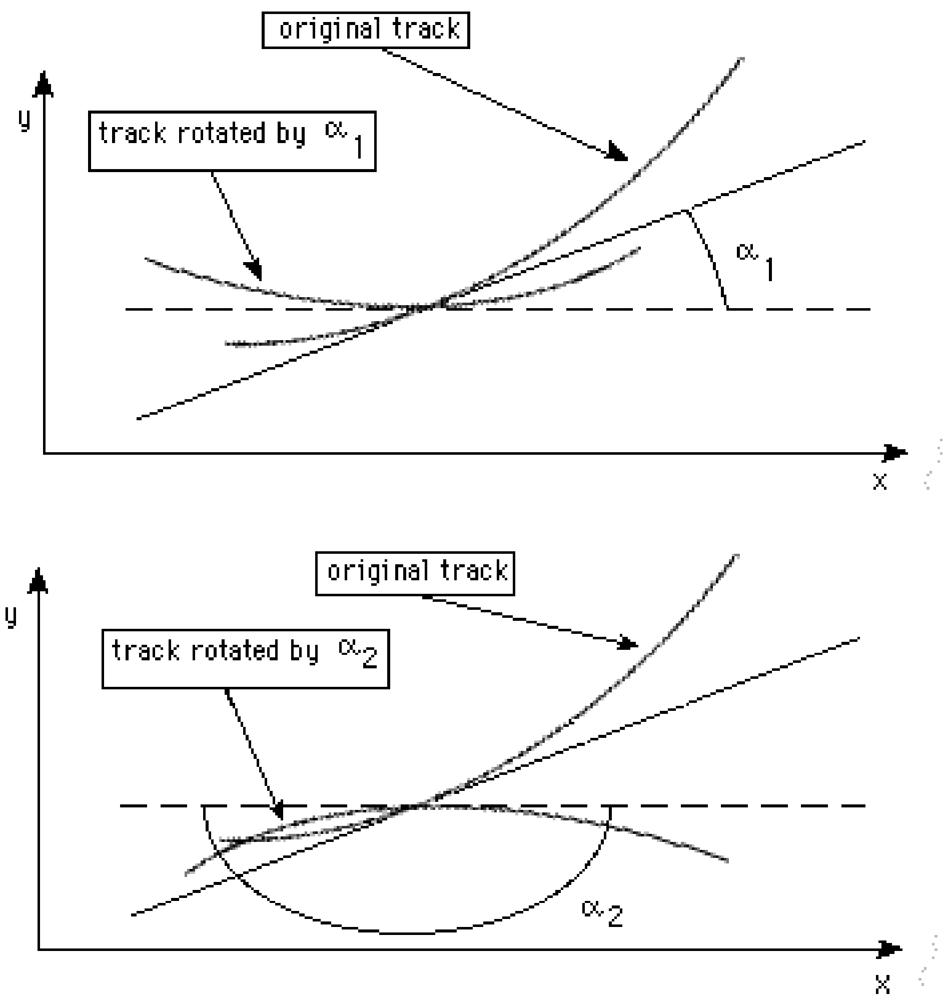

Figure 2.9: Two rotations with the same $|\sin(\alpha_{rot})|$, but with different curvature signs.

$$sign\left(\cos^2\alpha - \sin^2\alpha\right) = sign\left(\left[x^2\right]-\left[y^2\right]\right) \quad (2.2.17).$$

Equation 2.2.17 helps in the selection between the solutions u_1 and u_2, but the sign of the expression $\sin(\alpha_{rot})$ is not yet fixed. Figure 2.9 shows an example of two rotations that give the save value of $\sin(\alpha_{rot})$ but with opposite signs.

Since the interaction point is inside the detector and particles travel out from the center, all of these transformations leave the sign of a particle undefined. The upper or lower part of the circle can still represent both negatively and positively charged particles. In order to eliminate this ambiguity, it is required that the new x-axis must point outward from the beginning of the coordinate system ,or:

$\cos\alpha\left[x_{old}\right] + \sin\alpha\left[y_{old}\right] > 0$ (2.2.18).

With this condition, all of the lower parts of the circle represent one sign and all of the upper parts represent the opposite sign. Finally, x_i and y_i are scaled in such a way that $[x^2+y^2]=1$.

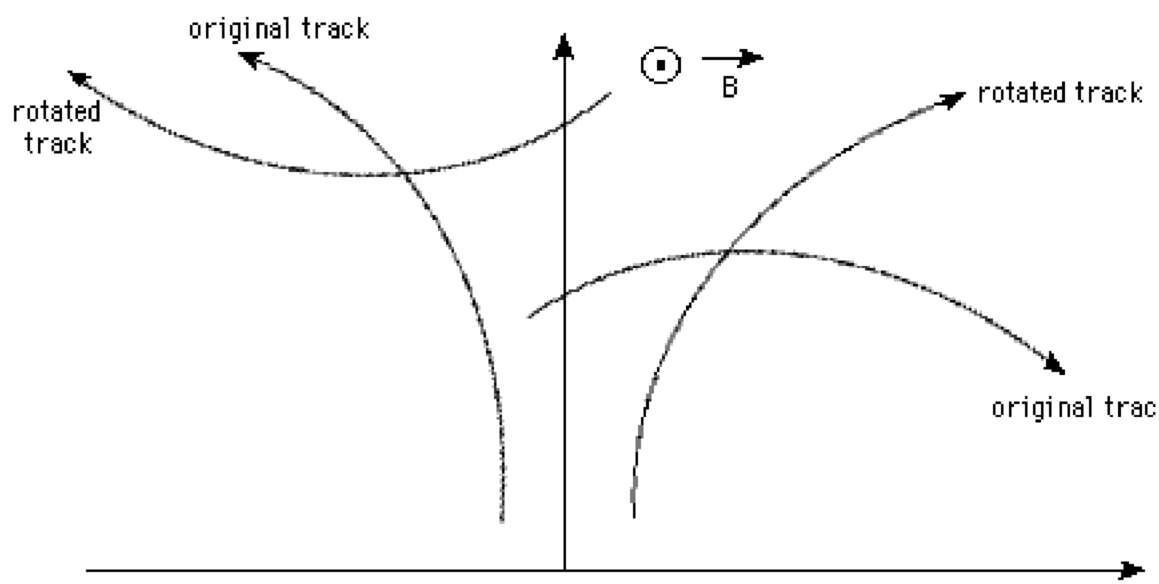

Figure 2.10: Rotation for the proper identification of the charge of a particle.

All of these transformations are performed in the subroutine UTCMAT. In this routine, the following definitions are used:

$RSCALE = \sqrt{\frac{\left[x_{old}^2 + y_{old}^2\right]}{[n]}}$ (2.2.19), where x_{old} and y_{old} are the translated and rotated coordinates. Also:

$$XXAV = \frac{\left[x^2\right]}{[n]} \equiv R, \quad YYAV = \frac{\left[y^2\right]}{[n]} \equiv P, \quad XRRAV = \frac{\left[x\left(x^2 + y^2\right)\right]}{[n]} \equiv Q,$$

$$YRRAV = \frac{\left[y\left(x^2 + y^2\right)\right]}{[n]} \equiv S, \qquad \text{and } RRRRAV = \frac{\left[\left(x^2 - y^2\right)^2\right]}{[n]} \equiv T \text{ (2.2.20).}$$

where (x,y) are the rotated, translated, and scaled coordinates. Equations 2.2.13 can be rewritten in the new coordinate system as follows:

$$-\frac{\left[x\left(x^2 + y^2\right)\right]}{[n]} + a\left(1 + \frac{2\left[x^2\right]}{[n]} - \gamma\right) = 0$$

$$-\frac{\left[y\left(x^2 + y^2\right)\right]}{[n]} + b\left(1 + \frac{2\left[y^2\right]}{[n]} - \gamma\right) = 0$$

$$\frac{\left[\left(x^2 + y^2\right)^2\right]}{[n]} - 2a\frac{\left[x\left(x^2 + y^2\right)\right]}{[n]} - 2b\frac{\left[y\left(x^2 + y^2\right)\right]}{[n]} - \gamma^2 = 0.$$

Here,

$$a = \frac{a_{old}}{RSCALE}, \; b = \frac{b_{old}}{RSCALE}, \text{ and } \gamma = \frac{\gamma_{old}}{RSCALE}.$$

If the symbols defined in Equations 2.2.20 are used and $(1-\gamma)/2$ is replaced by ω, these equations can be rewritten as:

$-Q+2a(\omega+R)=0$, $-S+2b(\omega+P)=0$, and $T-2aQ-2bS-(1-2\omega)^2=0$.

If a and b are calculated from the first two of these equations and replaced in the third, the following expressions can be obtained for ω:

$$2a = \frac{Q}{\omega + R},\ 2a = \frac{S}{\omega + P},\ \text{and}\ T = \frac{Q^2}{\omega + R} + \frac{S^2}{\omega + P} - (1 - 2\omega)^2.$$

These items can be regrouped in terms of powers of ω:

$$T(\omega + R)(\omega + P) - Q^2(\omega - P) - S^2(\omega + R) - (1 - 2\omega)^2(\omega + P)(\omega + R) =$$
$$-4\omega^4 + \omega^2(T + 3 - 4RP) + \omega\left(T - 1 - Q^2 - S^2 + 4RP\right) + RP(T - 1) - Q^2P - S^2R$$

In the subroutine UTCFIT, the following assignments are made:

C4=-4, C2=4+(T-1)-4RP, C1=(T-1)-Q^2-S^2+4RP, and C0=(T-1)RP-Q^2P-S^2R.

This equation is solved in the subroutine UTCFIT after replacing ω with -ω. The Newton-Raphson method is applied to solve this equation. The value of ω is directly connected to the chi-square value of the fit. With the denotation:

$$\omega' = -\omega,\ \omega' = \frac{\gamma - 1}{2},\ WSUM = \sum_{i=1}^{n} \frac{1}{w_i},\ \text{and}\ \gamma = \frac{\gamma_{old}}{RSCALE},$$

the following estimation can be obtained:

$$\omega' \times WSUM \times RSCALE^2 = 0.5 \times \gamma_{old} \times WSUM - 0.5 \times RSCALE^2 \times WSUM =$$
$$\sum_{i=1}^{n} \frac{R^2 - a^2 - b^2 - x_i^2 - y_i^2}{2w_i} = \sum_{i=1}^{n} \frac{R^2 - \left[(x_i - a)^2 + (y_i - b)^2\right]}{2w_i} \quad (2.2.21).$$

On the other hand, if the chi-squared value is estimated as:

$$\chi^2 = \sum_{i=1}^{n} \frac{\left(R - \sqrt{(x_i - a)^2 + (y_i - b)^2}\right)}{w_i} =$$
$$\sum_{i=1}^{n} \frac{R^2 + (x_i - a)^2 + (y_i - b)^2 - 2R\sqrt{(x_i - a)^2 + (y_i - b)^2}}{w_i},\ (2.2.22)$$

and the following is modified:

$$\sqrt{(x_i - a)^2 + (y_i - b)^2} = R\sqrt{\frac{R^2 + (x_i - a)^2 + (y_i - b)^2 - R^2}{R^2}} =$$
$$R\sqrt{1 + \frac{(x_i - a)^2 + (y_i - b)^2 - R^2}{R^2}}$$

and

$$\alpha = 1 + \frac{(x_i - a)^2 + (y_i - b)^2 - R^2}{R^2}.$$

Replacing these in Equation 2.2.22 yields the following expression:

$$\chi^2 = \sum_{i=1}^{n} \frac{2R^2 + \alpha R^2 - 2R^2\sqrt{\alpha + 1}}{w_i}.$$

If the chi-square value is optimized such that α is close to 0, then $\sqrt{\alpha + 1}$ can be replaced by the first few terms of its Taylor expansion, and the following is obtained:

$$\chi^2 = \sum_{i=1}^{n} \frac{2R^2 + \alpha R - 2R^2\left(1 + \frac{\alpha}{2} - \frac{\alpha^2}{8}\right)}{w_i} = \sum_{i=1}^{n} \frac{\alpha^2 R^2}{4w_i} =$$
$$\sum_{i=1}^{n} \frac{\left[(x_i - a)^2 + (y_i - b)^2 - R^2\right]^2}{4w_i R^2} = \sum_{i=1}^{n} \frac{R^2 - \left((x_i - a)^2 + (y_i - b)^2\right)}{2w_i} \quad (2.2.23)$$

which is identical to Equation 2.2.21.

Section 2.2.2: Line Fitting

Lines are fitted in the subroutine LSWLF. The linear fit is a simple least squares fit to a line of the form y=ax+b using linear regression.

First, the weighted sums of all of the points on the track are calculated as follows:

$$S = \sum_{i=1}^{n} w_i,\ S_x = \sum_{i=1}^{n} w_i x_i, S_y = \sum_{i=1}^{n} w_i y_i,\ S_{xx} = \sum_{i=1}^{n} w_i x_i^2,\ S_{yy} = \sum_{i=1}^{n} w_i y_i^2,\ S_{xy} = \sum_{i=1}^{n} w_i x_i y_i$$

The determinant, D, is defined as: $D = S \bullet S_{xx} - S_x \bullet S_x$ (2.2.25). From this, the slope and intercept of the line is calculated as follows:

$$a = \left(S \bullet S_{xy} - S_x S_y\right)/D,\ b = \left(S_y \bullet S_{xx} - S_{xy} S_x\right)/D \quad (2.2.26).$$

The errors and chi-square value of the fit are calculated using:

$$\sigma_a = S/D,\ \sigma_b = S_{xx}/D, \text{ and } \chi^2 = \sum_{i=1}^{n} w_i\left[y_i - Ax_i - b\right]^2 \quad (2.2.27).$$

Section 2.2.3: Track Parametrization

The trajectory of a particle in a static uniform magnetic field with a field vector parallel to the z-axis is a helix with an axis along the field lines as shown in Figure 2.10. The helix can be paramtrized as follows:

$$x(s) = x_0 + R_H\left[\cos\left(\Phi_0 + \frac{hs\cos\lambda}{R_H}\right) - \cos\Phi_0\right]$$

$$y(s) = y_0 + R_H\left[\sin\left(\Phi_0 + \frac{hs\cos\lambda}{R_H}\right) - \sin\Phi_0\right]$$

$$z(s) = z_0 + s\sin\lambda \quad (2.2.24)$$

where:

- s is the path length along the helix. It increases when moving in the direction of the particle's momentum vector.

- (x_0, y_0, z_0) are the coordinates of the starting point of the helix. This is where $s=s_0=0$.

- λ is the slope of the helix, also referred to as the dip angle. This is equal to the arcsin(dz/ds) where $-\pi/2<\lambda\leq\pi/2$.

- R_H is the radius of the helix. This is given by the expression:

$$R_H = \frac{P\cos\lambda}{|qB\kappa|} = \frac{1}{\left(\frac{1}{p_t}\right)} \times 0.001499$$

with the standard STAR magnetic field.

- κ is the conversion factor. If the radius is in units of centimeters, the magnetic field is in units of kiloGauss, and the momentum is in units of GeV, κ=0.00003.

- B is the value of the magnitic field parallel to the z-axis.

- q is the charge of the particle.

- h is the sense of rotation of the projected helix in the x-y plane. This is equal to sign(qB)=sign($d\psi/ds$).

- ψ is the azimuthal angle of the momentum.

- z is the polar axis parallel to the helix axis.

- Φ_0 is the azimuthal angle of the starting point of the helix in cylindrical coordinates with respect to the helix axis. This is equivalent to $\Phi_0=\Psi-h\pi/2$.

- Ψ is $\arctan(dy/dz)_{s=0}$. This is the azimuthal angle of the track direction at the starting point of the helix.

For a more detailed description of the helix parameters, see Reference [6].

The circle fit described in the Section 2.2.1 returns the center of the fitted circle as the coordinates (x_c, y_c) along with the value of R_H. The linear fit described in Section

2.2.2 returns the values of z_0 and $\tan(\lambda)$. In order to calculate the parameters to be placed into the tptrack table, the starting point of the helix must be determined. In order to facilitate track matching to the SVT, it is logical to select a point closest to the SVT. Therefore, the azimuthal angle in the helix coordinate system for the point closest to the TPC inner field cage is evaluated as follows:

$$\Phi_0 = \arctan\left(\frac{y_1 - y_c}{x_1 - x_c}\right)$$

where $(x_1.y_1)$ are the coordinates of the closest TPC point. A point on the fitted track is selected by evaluating:

$x_0 = x_c + R_H \cos \Phi_0$ and $y_0 = y_c + R_H \sin \Phi_0$.

The elements used in the tptrack table are then evaluated:

$r_0 = \sqrt{x_0^2 + y_0^2}$, $\phi_0 = \arctan\left({}^{y_0}\!/_{x_0}\right)$, $\Psi = \Phi_0 + {}^{h\pi}\!/_{2}$,

$p_t = R_H \times |qB\kappa|$, $p_z = p_t \tan \lambda$, and $p = \sqrt{p_t^2 + p_z^2}$ (2.2.25).

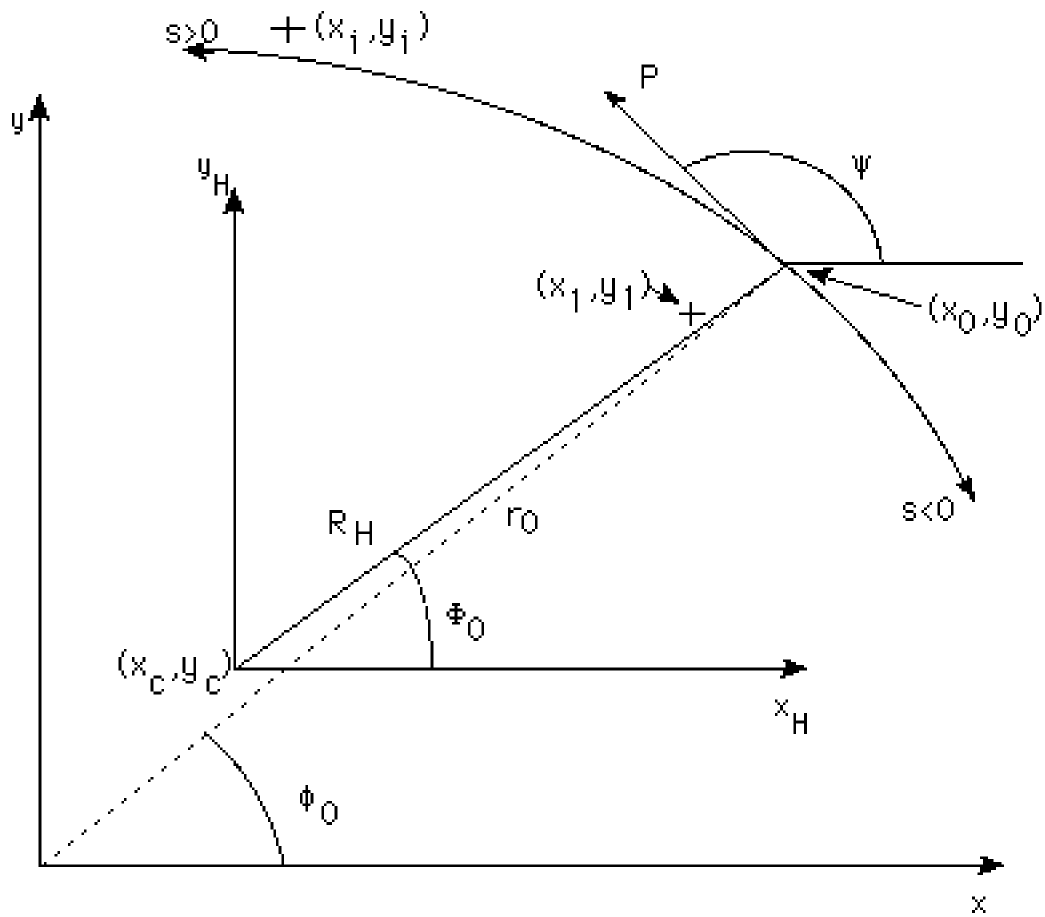

Figure 2.11: projection of a helix on the x-y plane. The points (x_i,y_i) are the points on the fitted track. Other symbols are described in the text of this section.

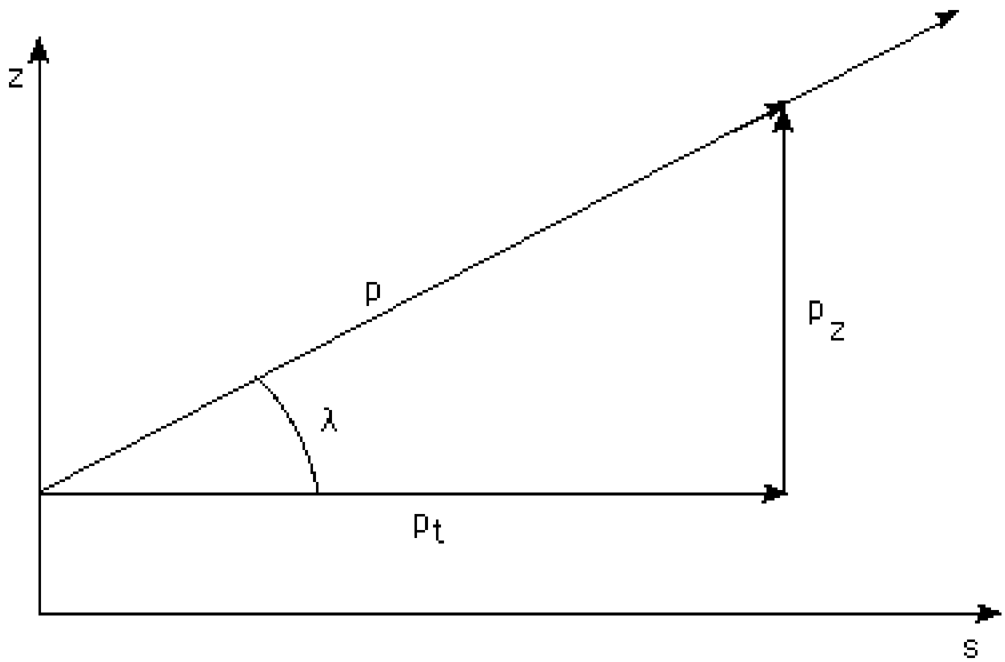

Figure 2.12: Projection of a helix in the s-z plane.

Section 2.3: Tracking Demonstration

In order to demonstrate the operation of the different phases of the tracking algorithm, this section will trace the evolution of the formation of two separate tracks, one with a transverse momentum of 150 MeV/c (which barely remains contained within the TPC) and the other with a transverse momentum of 1 GeV/c. These tracks were created by generating a simple ASCII file that was then read into GEANT. GEANT tracked these particles and returned a set of hit coordinates for each row crossing of the tracks. The GEANT output file was then fed into a TAS executable that ran the mct module in order to copy the GEANT output banks into TAS tables, followed by the tfs module, or TPC Fast Simulator, followed by the tpt module. The input set of hit coordinates projected onto the x-y plane as returned from tfs are plotted in Figure 2.13 for the two tracks to be used in this demonstration.

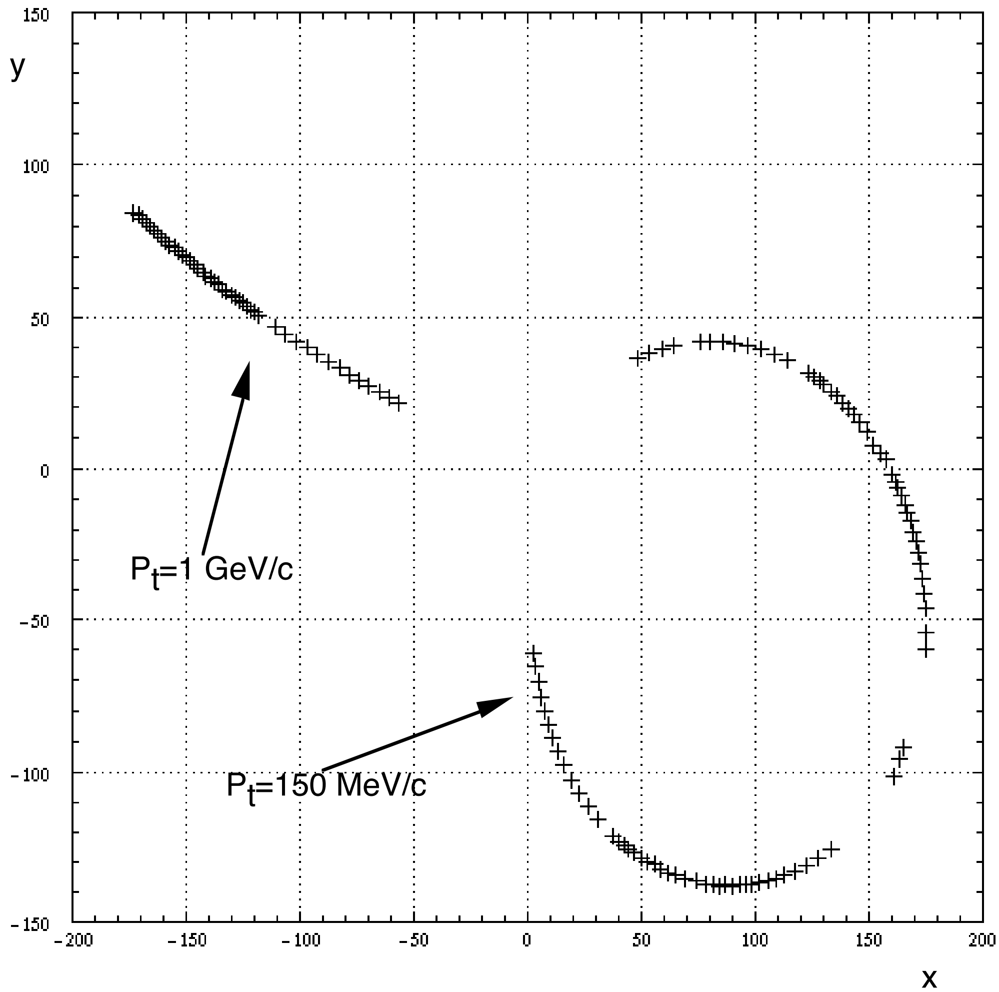

Figure 2.13: The input points returned from tfs to be processed by tpt in this demonstration. The points have been projected onto the x-y plane.

The first step of the track finding process is that of root finding. The tracking program begins a search on the outside rows of the TPC and works inward in order to find a set of three-point links from which to form segments. For the 1 GeV/c track, which is a relatively straight one, the three points in the three outermost rows of the TPC are chosen as the root as shown in Figure 2.14. The problem becomes more complicated for the 150 MeV/c track since it does not pass the root-forming cuts in the outer row. Instead, the cuts are not passed until further within the TPC and the roots are created there. As a result, the tracking algorithm actually finds four separate roots for this one input track as shown in Figure 2.15.

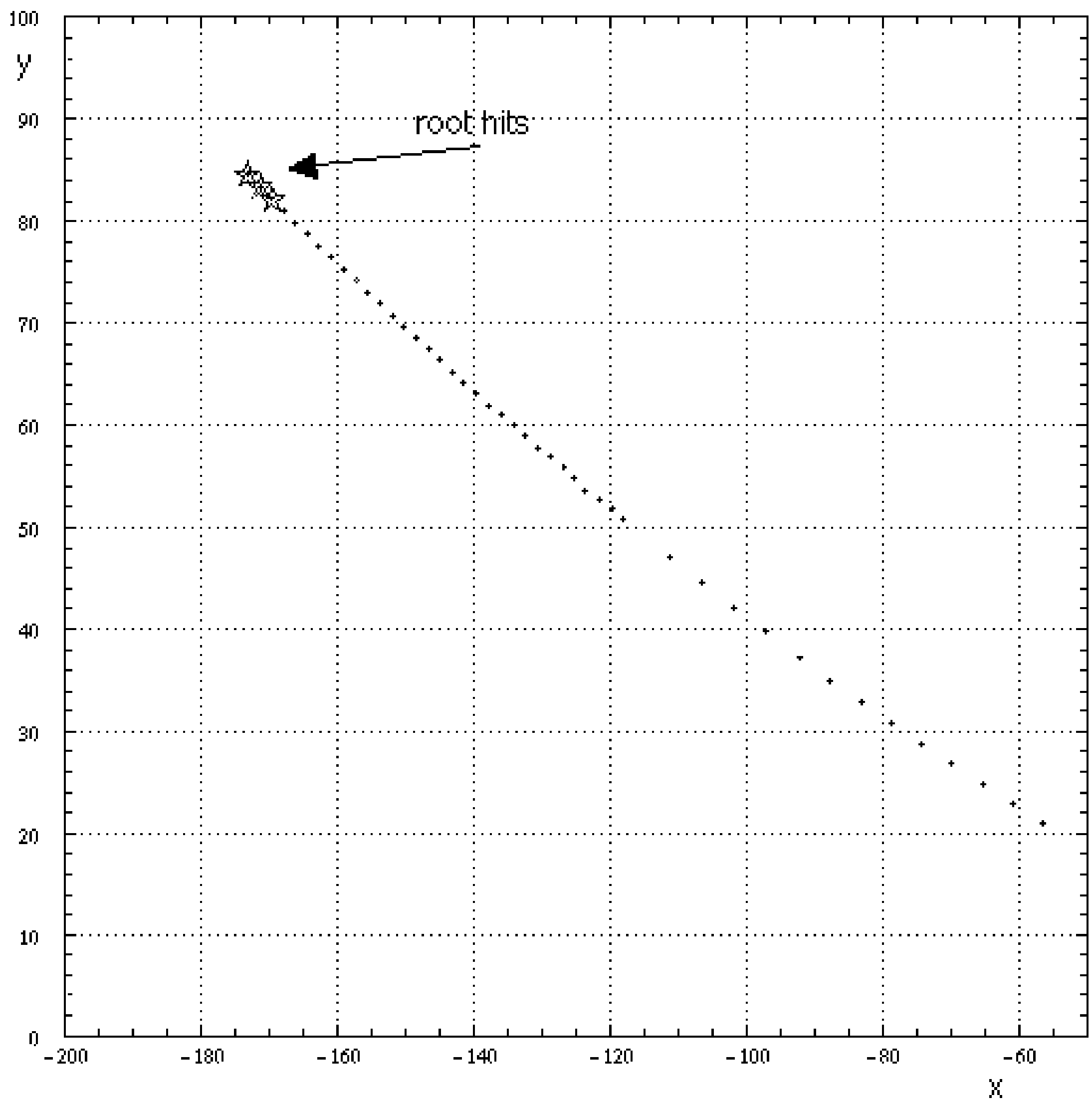

Figure 2.14: The dots are the hits that will eventually make up this track. The stars are the points that are identified as the root of the track. Notice the difference in separation of the points between the inner sector and the outer sector.

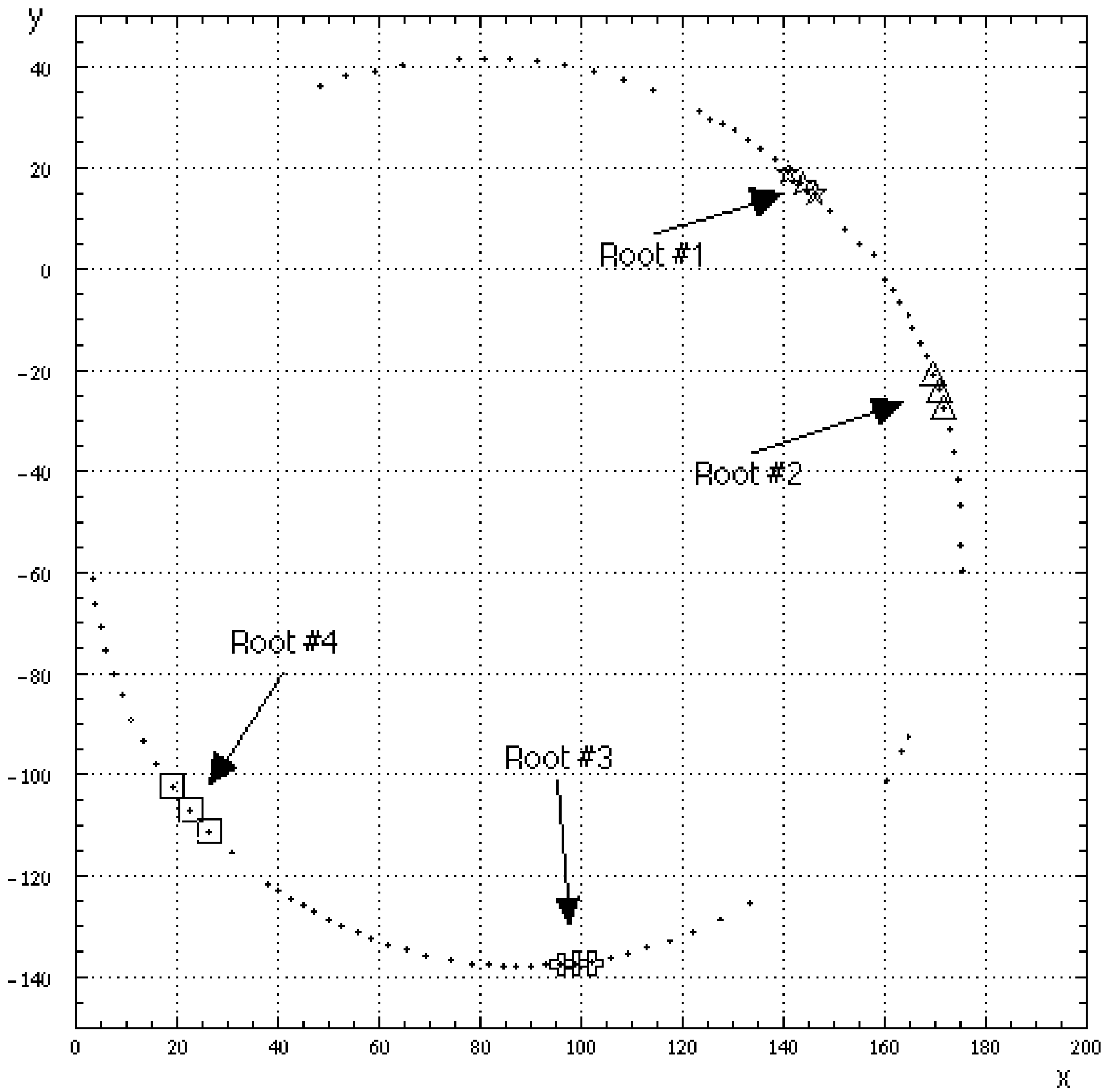

Figure 2.15: The root formation step results for the 150 MeV/c track. Here four roots have been found for the one input track. The input hits for this track are shown as dots. The other symbols are the roots for the four segments that will be found in the next step.

The next step of the tracking process is that of segment formation. This demonstration has the conformal mapping method activated. For each root found, the root is projected inwards in the TPC towards the event vertex. If hits along the way fall within tolerances of the projection, they are added to the segment. Note that once a root is found, the segment formation process is performed on it immediately before the next root is found. This explains the seemingly strange selection of the roots for the 150 MeV/c track.

The segment formation step associates all 45 of the possible points (one for each pad row crossing of the track) for the 1 GeV/c track. Therefore, track reconstruction is complete for that track after this step. For the 150 MeV/c track, four roots were found, so four segments were formed. The points associated with each segment are illustrated in Figure 2.16 below.

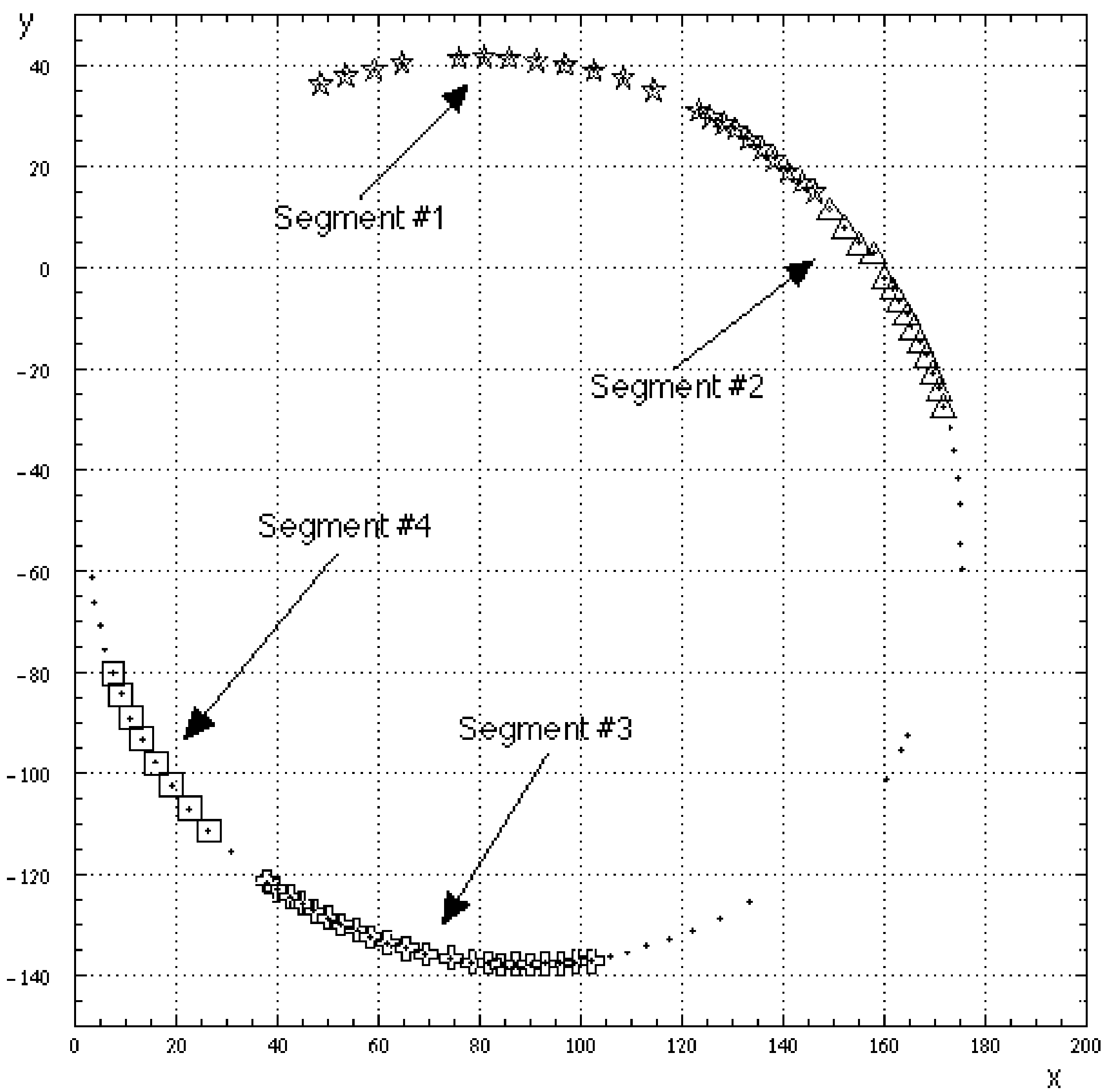

Figure 2.16: The results of the segment formation phase of the tracking on the 150 MeV/c track. The symbols differentiate the segments. The dots are the input points that still have not been associated with segments.

Note that there are still some points in the outer pad rows that have not been associated with a segment. This is due to the fact that the roots were extended inwards only during the segment formation process. In order to pick these up, the next phase of the tracking algorithm, the segment extension is applied. This uses a helix model projected both inwards and outwards from the existing segment in order to pick up any further points that lie within tolerances of the projection. The result of the segment extension on the 150 MeV/c track is illustrated in Figure 2.17.

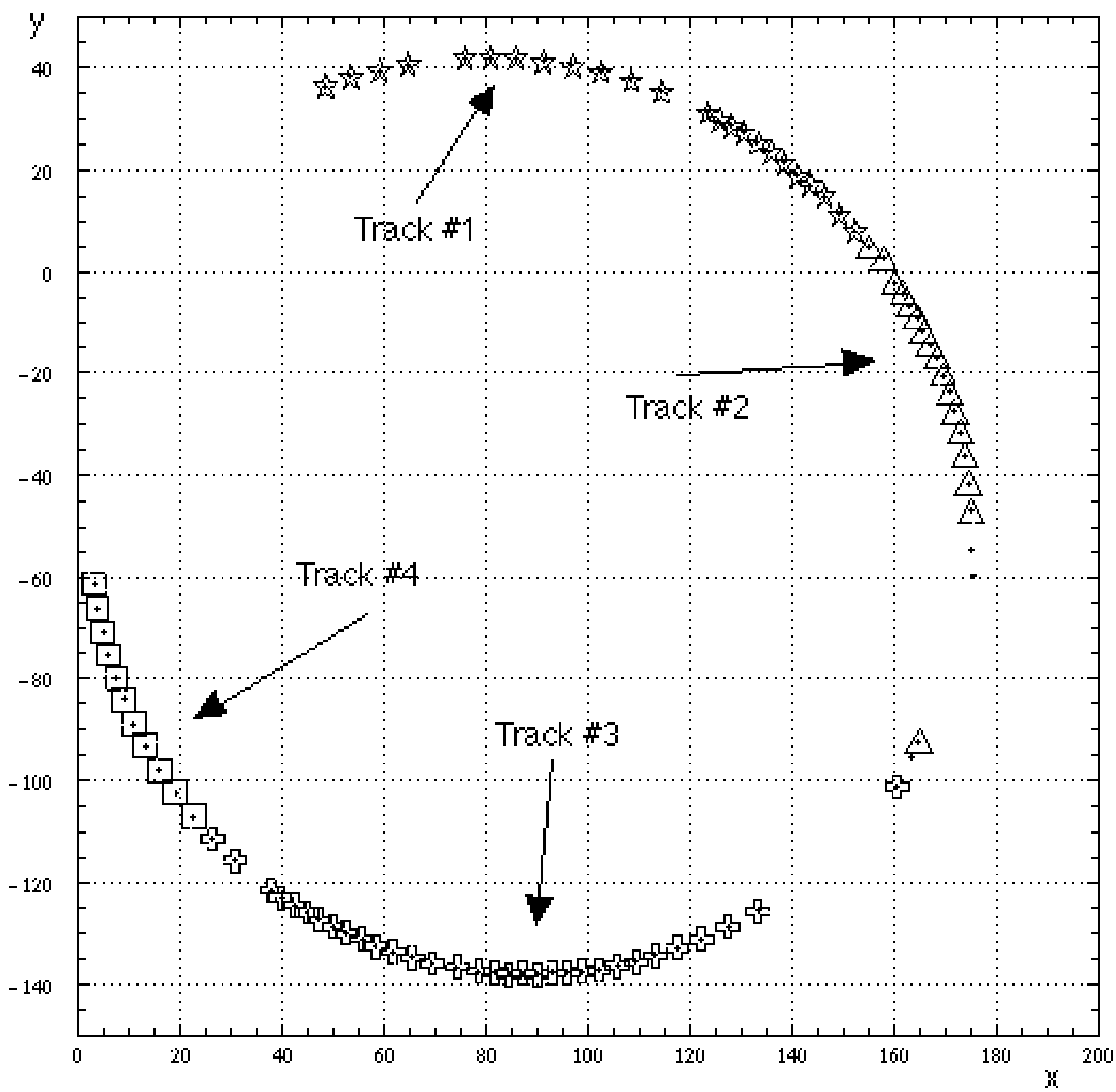

Figure 2.17: The symbols differentiate the segments formed from the 150 MeV/c track after the segment extension phase of the tracking algorithm.

As shown in Figure 2.17, virtually all of the available points have been associated with a segment. The problem is that there are four segments. This problem is handled by the helix merging phase of the tracking algorithm. The helix merging compares the helix parameters obtained from helix fits of the points associated with each segment. If they agree within the tolerances, the segments are merged into a single track segment. Therefore, an optimized helix merging phase associates all four of these segments into a single track.

CHAPTER 3: TPT USER'S GUIDE

This chapter provides useful and essential information for those folks interested in running the STAR TPC tracking package. The tracking package consists of two modules: the tpt module that performs the track reconstruction and estimates the track helix parameters, and the tte module that performs an evaluation of the tpt module performance. This chapter explains how to build, run, and control the tracking and tracking evaluation packages. It also explains the output tables of these packages.

The tracking program works with the reconstructed TPC space-points. Based on local correlations between the points, tracks are formed and the tracks are fitted to a helix. The following sections provide information about how to control the flow of the tpt package. Also included is a description of the helix parameters and documentation on the output of tpt.

Section 3.1. TPT Usage

The tpt package can be run in two modes. These two modes are controlled by the parameter tptpar.mctest. Setting mctest to 0 will result in full realistic tracking using all of the facilities of the tracking algorithm described in Chapter 2. In order to run in this mode, the following tables are needed:

tptpar = tracking parameters.
pad_plane = description of the pad plane geometry
tphit = list of the TPC space-points to be considered by the tracking code.
tptrack = output from tpt.

To fill these tables, either the output from the TPC Fast Simulator (tfs) or the hit finder (tcl) must be available. If output from tfs is used, it is important to remember to add the filter tfs_filt.f (see Ref. [2]) on the tphit table in order to remove merged or deleted hits. An example of the calling sequence for this filter from a TAS macro (.kumac) file is given below:

```
TAS> def/filt add tpt tpt_filt.f(tptrack.flag)
TAS> def/filt on tpt
```

TAS> def/filt sho

This filter can be found in the /wrk section of the tpt directory in the STAR library. It is not necessary to use this filter if the evaluation is to be called next.

In the second tpt running mode (tptpar.mctest=1), tracks are formed using the Monte Carlo information and the resulting tracks are fitted to a helix. In this mode, an additional table (mhits_tpc = GEANT output for the TPC) must be available. The Module Interface Definition (or .mid) file (see Ref. [7]) for the tpt module is as follows:

```
#
table_lib mct
table_lib tpg
table_lib tcl
table_lib tpt
#
module tpt tpt_all_main
#
module_table    tpt    tptpar       update    1
module_table    tpt    pad_plane    read      2
module_table    tpt    tphit        update    3
module_table    tpt    mhits_tpc    read      4
module_table    tpt    tptrack      write     5
```

The tracking program consists of the following steps: root formation, segment formation with conformal mapping, segment formation with linear extrapolation, segment extension across sector boundaries, and helix merging. The last four steps can be turned on or off vea the appropriate tptpar parameter settings (see section 3.1.2.).

Additionally, intermediate steps of the track formation process can be displayed by the event visualization programs. This feature can be enabled by setting the tptpar.display parameter to a value described in Section 3.1.2. If the display parameter is set, an input file for the event visualizer is generated. To synchronize output from the tracking with the event display program and to allow the user to study intermediate steps of the tracking process, the READ statements embedded in the code are activated with this option. In order to resume execution of the program, the user must press <Enter> on the keyboard.

If more detailed information for debugging purposes is needed, a set of debug flags is provided. They turn on the printing of additional information that allows the user to follow track formation. In order to allow the user to follow printed information, READ statements are again used and when encountered, the user must hit <Enter> on the keyboard. The debug flags are also documented in Section 3.1.2.

Section 3.2. TPT Control Parameters

The tpt package is controlled through the tptpar parameter table. These parameters define the program flow, tolerances, and level of message printing. A detailed description of the use of these parameters can be found in the description of the tracking algorithm in Chapter 2. Below is a short description of each of the tptpar parameters:

tptpar.g1 = the maximum acceptable row gap between the first two points on a track.

tptpar.g2 = the maximum acceptable row gap between the second and the third point on a track.

tptpar.irow_root_beg = the outermost row in which to start track root formation.

tptpar.irow_root_end = the innermost row in which to end track root formation.

tptpar.irow_seg_end = the innermost row in which to end the segment extension from the roots formed in the first stage of the track reconstruction process.

tptpar.minfit = the minimum number of hits on a track that is acceptable for helix fitting.

tptpar.mxtry = the maximum number of attempts to remove an outlier and improve the quality of the track helix parameters.

tptpar.nhol = the number of consecutive hits (rows) missing on a track before the track segment formation process is terminated for that track.

tptpar.delrlm = the maximum distance in the bend plane between the first two points on a track in centimeters.

tptpar.delzlm = the maximum distance in the r-z plane by which the second point on a track may lie from a straight line between the first point on the track and the vertex, in centimeters.

tptpar.drlm1 = the limit imposed on the distance in the bend plane between the third point on a track and the prediction of the track's crossing at the center of the row where the third point is to be found. A straight line is used as the track model and the track parameter predictions are evaluated using the first two points. This is measued in the units of the standard deviation of the signal width.

tptpar.dzlm1 = the limit imposed on the distance in the r-z plane between the third point on a track and the prediction of the track's crossing at the center of the row where the third point is to be found. A straight line is used as a track model and the track prediction is evaluated using the first two points of the track. This is measured in units of the standard deviation of the amplifier response.

tptpar.dzplim = the maximum distance between a candidate point that is to be assigned to a track and the prediction of the track's position on that row in the r-z plane during segment formation. This is measured in units of the standard deviation of the amplifier response.

tptpar.drplim = the maximum distance between the candidate point that is to be assigned to a track and the track's prediction in the row of that point in the bend plane during segment formation. This is measured in units of the standard deviation of the pad response.

tptpar.drmax = the maximum tolerance in the bend plane for assigning additional points to previously formed track segments. This is measured in centimeters.

tptpar.dzmax = the maximum tolerance in the r-z plane for assigning additional points to previously formed track segments. This is measured in centimeters.

tptpar.drstr = the tolerance in the bend plane for assigning additional points to previously formed track segments. This is measured in units of the standard deviation of the pad response. If the distance measured in centimeters is larger than tptpar.drmax, then this is set to tptpar.drmax/σ.

tptpar.dzstr = the tolerance in the s-z plane for assigning additional points to previously formed track segments. This is measured in units of the standard deviation of the amplifier response. If the distance measured in centimeters is larger than tptpar.drmax, then this is set to tptpar.drmax/σ.

tptpar.mxweig = the limit on the number of hits on a pad after which the hit is disqualified as the the starting point for a track.

tptpar.prob(1) = the probability cut for the circle fit. If this is negative, no outlier removal is performed.

tptpar.prob(2) = the probability cut for the linear fit. If this is negative, no outlier removal is performed.

tptpar.zslim = the limit between the z-coordinate of the first and the last point on a track. If this distance is smaller than this limit, the track is fitted in the z-s plane instead of the s-z plane.

tptpar.mctest = allows the user to select full, realistic tracking with mctest=0, or an ideal track fit of hits grouped using the Monte Carlo information with mctest=1.

tptpar.debug(1) = a flag that controls the debug printing at the highest level. When set, this parameter will supply information on CPU time for the entry and exit points of the major code segments.

tptpar.debug(2) = a flag that controls debug printing for the segment extension process.

tptpar.debug(3) = a flag that controls debug printing for the root finding process.

tptpar.debug(4) = not used.

tptpar.debug(5) = a flag that controls debug printing for the segment formation process.

tptpar.debug(6) = a flag that controls debug printing for the outlier removal process.

tptpar.debug(7) = a flag that controls printing for the program flow when the tptpar.mctest flag is set to 1.

tptpar.debug(8) = not used.

tptpar.debug(9) = a flag that controls printing of accounting problems, or inconsistencies in assigning the points to tracks.

tptpar.display = an option that allows the user to turn on generation of output files to be read by an event display. (1=stargl, 2=cos, 3=cos innego)

tptpar.form_uv = toggles the segment formation using conformal mapping process.

tptpar.form_uv_p = toggles the segment formation using conformal mapping process with a fixed position of the interaction point (e.g., the event vertex).

tptpar.form_xy = toggles the segment formation using linear extrapolation process.

tptpar.extend = toggles the segment extension process.

tptpar.merge = toggles the spiral merging process.

tptpar.nzslic = the number of subdivisions along the z-axis with which to divide the TPC in order to speed up search processes.

Section 3.3. TPT Output

Information about the reconstructed tracks is stored in the tptrack table. A description of the variables contained within this table is given below:

tptrack.flag = Status flag. If this is negative, the track should be discarded.

tptrack.hitid = id of the hit where the track parameters are evaluated. This is the primary key of that hit in the tphit table.

tptrack.id = the track id. This is the primary key for this table.

tptrack.nfit = the number of points included in the momentum fit.

tptrack.nrec = the number of points assigned to the reconstructed track.

tptrack.q = the charge of the particle.

tptrack.chisq(1) = chi-square value for the momentum fit in the bend plane.

tptrack.chisq(2) = chi-square value for the momentum fit in the s-z plane.

tptrack.cov(15) = covariance matrix for the track fit defined as follows:

$1/p_t$	ϕ^0	r^0	$\tan\lambda$	z^0	
$1/p_t$	c(1)				
ϕ^0	c(2)	c(3)			
r^0	c(4)	c(5)	c(6)		
$\tan\lambda$	c(7)	c(8)	c(9)	c(10)	
z^0	c(11)	c(12)	c(13)	c(14)	c(15)

tptrack.dedx(1) = average energy loss per unit length (GeV/cm).

tptrack.dedx(2) = the error on the average energy loss per unit length.

tptrack.invp = the inverse of the fitted transverse momentum ($1/p_t$). Remember, this the transverse momentum, not the total momentum.

tptrack.psi = the azimuthal angle of the transverse momentum at the first point on the track (ψ).

tptrack.phi0 = the azimuthal angle of the first point on the track (ϕ^0).

tptrack.r0 = the radial coordinate of the first point on the track (r^0).

tptrack.tanl = the tangent of the track's dip angle measured from the bend plane ($\tan\lambda$).

tptrack.z0 = the z-coordinate of the first point on the track (z^0).

A more detailed description of the track parameters and what they mean is given in Section 2.2.3.

The tracking code also fills the appropriate tphit values that identify which track the hit belongs to. The value tphit.track is encoded with the track number along with its position along the track. The latter value has been installed to facilitate the event displays. The formula for this encoding is tphit.track = (track number)x100+(hit position on track). The hit position is numbered from the outside of the TPC to the inside. This value is set to zero if the hit is not associated with a track.

CHAPTER 4: TRACKING EVALUATION

Evaluation of the tracking performance is performed by the tte module. This module compares the Monte Carlo tracks that served as input to the tpt module to the tracks reconstructed by the tracking program. This chapter will contain a description of the evaluation philosophy embedded within a description of each subroutine contained within tte in Section 4.1. A description of the control of the evaluation, the output of the evaluation, and how to interpret the results is included in Section 4.2. Section 4.3 contains a guide on how to use the tte module within a TAS executable.

Section 4.1. Track Evaluation Strategy and Subroutines

This section will detail a subroutine-by-subroutine description of the tte module. By studying this section along with Section 4.2, a complete picture of the tte evaluation philosophy should be painted. The philosophy is actually a simple one, but for those interested in the gory details, a look at the code can give insight into the accounting nightmare that must be dealt with here.

TTE_MAIN: This is the main routine of tte from which all other subroutines are called. The flow of the evaluation is also controlled here. It begins by calling various subroutines that setup lookup lists of the hits and tracks and their relationships (via the subroutines tte_list_mchits, tte_list_trk, tte_list_mpart, tte_list_mvert, tte_list_mkine, tte_trk_pnt, tpt_hit_pnt, and tte_trk_mc). This routine then proceeds to loop over all of the reconstructed tracks. For each reconstructed track, it finds all of the Monte Carlo information about the first point assigned to that track. Tracks that do not have a good flag (tptrack.flag>0) are skipped. The dominant Monte Carlo track contributor to the reconstructed track is determined as the one with the highest fraction of hits on the reconstructed track assigned to it. This routine also calculates the production or decay vertex for the track from the Monte Carlo information along with the particle momentum at that vertex. For tracks not originating from the event vertex, the track is refitted with the true vertex. These functions are performed with help from other subroutines (tte_trk_res, tte_mctrk, tte_mctrk_mkine, tte_mctrk_rectrk, and tte_mctrk_pair).

Finally, this subroutine keeps track of the CPU time expended in each step of the evaluation process.

The subroutines listed below are called by tte_main at the beginning in order to set up the accounting information relating tracks and hits, both reconstructed and Monte Carlo:

TTE_LIST_TRK: This subroutine associates reconstructed tracks with their id value. For every track, it generates an entry in the array list_trk(tptrack.id) which is the TAS table row number for that track in tptrack.

TTE_TRK_PNT: This subroutine sets up pointers to the hits contained within every reconstructed track. It begins by looping through all of the hits. Every hit that was assigned to a track has a non-zero value of tphit.track which is coded as tphit.track=(track number=itrk)*1000+(position of the hit on the track=ient). This information is decoded and the array trk(itrk).ipnt(ient) is filled with the TAS table row number of the hit in the table tphit.

TTE_TRK_MC: Associates the Monte Carlo track contributions to a reconstructed track. First, all hits on every track are looped over. For each hit, its original Monte Carlo track number is extracted. The number of contributions from each Monte Carlo track extracted is counted for all hits in the reconstructed track. If a hit has been merged by the TPC Fast Simulator or cluster finder, all Monte Carlo track contributors to that hit are counted, but each contributor is only counted once per hit. This information is stored in the list_mctrcks array.

TTE_LIST_MPART: Establishes a relationship between the Monte Carlo particle id and the entry number for that particle in the mpart (list of particles) table. For every particle with an id of mpart.ipart, the array list_mpart(mpart.ipart) is used to store its position in the mpart table. In other words, list_mpart allows the user to find a particle using its id in the mpart table.

TTE_LIST_MVERT: Establishes the relationship between the Monte Carlo vertex and its mvert entry number through the array list_mvert(mvert.id) which allows the user to find a Monte Carlo vertex using its id in the mvert table. This is done for every particle with a vertex id of mvert.id.

TTE_LIST_MKINE: Sets up the correspondence between the Monte Carlo track id and its TAS row number in the mkine table through the array list_mkine(mkine.id), which allows the user to find a Monte Carlo track using its id in the mkine table.

TTE_LIST_MCHITS: Creates pointers to the Monte Carlo hits. This routine loops through all of the Monte Carlo hits. For every hit with a nonzero mhits_tpc.recdig value, the array list_mchits(recdig) is assigned the TAS table row number of the mhits_tpc table. If there is more than one Monte Carlo hit with a given recdig value, that entry of list_mchits is assigned a negative value and becomes a pointer into a location within the list_mchits array that contains the number of hits sharing that recdig value. Following this location in list_mchits is a list of pointers for those hits to their Monte Carlo mhits_tpc table entry. In other words, list_mchits allows the user to address Monte Carlo hits using the id of a hit that was reconstructed from them.

The subroutines below are called by tte_main during its execution to calculate various quantities:

TTE_TRK_RES: Calculates the residuals for the reconstructed tracks. For every point assigned to a reconstructed track, fitted values of the coordinates at each pad row are calculated and the difference between the fitted and actual hit coordinates are calculated as residuals. The chi-square value for every track is recalculated, also.

TTE_MCTRK: Finds Monte Carlo tracks that cross the TPC and counts the number of row crossings for that track. This routine loops over all of the Monte Carlo hits generated within the TPC. It finds their Monte Carlo track number and counts the number of occurrences of each track. It also checks whether or not the crossing survived the digitization process and counts the ones that have. It also keeps track of the number of hits on each track that have merged.

TTE_MCTRK_MKINE: Fills the tp_mctrk table with information about the Monte Carlo track. See the next section for more information about this table.

TTE_MCTRK_RECTRK: Fills the tp_mctrk table with information about the reconstructed track. See the next section for more information about this table.

TTE_MCTRK_PAIR: Fills the tp_ev_pair table with information about the reconstructed track segments. This table is described in the next section, and is filled by this routine only for Monte Carlo tracks which have been identified as the dominant contributors to more than one reconstructed track.

Section 4.2. Track Evaluation Input and Output

This section will describe the input, control, and output tables utilized by the tte module. The input tables that must be filled in order for tte to perform properly are tphit (the reconstructed TPC hit table), tptrack (the reconstructed TPC track table), mhits_tpc (the GEANT TPC hit, or row crossing, table), mkine (the GEANT kinematics bank table), mpart (the GEANT particle data bank table), and tptpar (the TPC tracking parameters table).

The tte module is controlled through the ev_cntrl table, whose elements are described below:

ev_cntrl.pair: toggles on or off the evaluation of track segments.

ev_cntrl.res: toggles on or off the residual calculation.

The tte module sends output to four different tables which will be described below. The largest, and most general, of these output tables is the tpeval table. This table contains one entry for every reconstructed track segment and contains the following elements:

tpeval.mcid: the Monte Carlo track identifier of the dominant contributor to this reconstructed track.

tpeval.nfit: the number of points that were included in the parameter fit for this reconstructed track.

tpeval.ngood: the number of points from the most significant Monte Carlo track contributor associated with this reconstructed track.

tpeval.nrec: the number of points assigned to this segment during the segment reconstruction procedure.

tpeval.qg: the charge of the most significant Monte Carlo contributor to this reconstructed track.

tpeval.qr: the reconstructed charge of the track.

tpeval.rtrk: the reconstructed track number. This is the primary key of the corresponding entry in the tptrack table.

tpeval.vid: the vertex id of this track.

tpeval.abpullav: the average pull, defined as the total residual divided by the weight.

tpeval.chisq: the chi-square value of the fit to this track.

tpeval.pid: the Monte Carlo particle id as assigned by GEANT for the most significant Monte Carlo contributor to this track.

tpeval.prob: the probability assignment for this track.

tpeval.ptg: the transverse momentum generated at the first point on the track.

tpeval.ptr: the transverse momentum reconstructed at the first point on the track.

tpeval.ptv: the transverse momentum generated (Monte Carlo) at the vertex for this track.

tpeval.pullav: the average pull. This is defined above.

tpeval.pullmx: the maximum pull.

tpeval.pxg: the x-component of the momentum generated (Monte Carlo) at the first point of the track segment.

tpeval.pxr: the x-component of the momentum reconstructed at the first point of the track segment.

tpeval.pxv: the x-component of the momentum generated (Monte Carlo) at the vertex for this track.

tpeval.pyg: the y-component of the momentum generated (Monte Carlo) at the first point of the track segment.

tpeval.pyr: the y-component of the momentum reconstructed at the first point of the track segment.

tpeval.pyv: the y-component of the momentum generated (Monte Carlo) at the vertex for this track.

tpeval.pzg: the z-component of the momentum generated (Monte Carlo) at the first point of the track segment.

tpeval.pzr: the z-component of the momentum reconstructed at the first point of the track segment.

tpeval.pzv: the z-component of the momentum generated (Monte Carlo) at the vertex for this track.

tpeval.qfact: the quality factor defined as the number of good points assigned to the track (ngood) divided by the total number of points on the track.

The trk_res table contains information on the residuals for every hit assigned to a track, and contains one entry for every one of these hits:

trk_res.trk: the track id of the track to which this hit was assigned. This is the primary key of the tptrack table for that track.

trk_res.sr: the residual for this hit in the bend plane.

trk_res.sz: the residual for this hit in the s-z plane.

The tp_ev_pair table contains information about the Monte Carlo tracks that are dominant contributors to more than one reconstructed track. This provides information about "fragmented" tracks output from the track reconstruction routine:

tp_ev_pair.dnrec1: the difference in the number of reconstructed hits between the two longest segments associated with the Monte Carlo track.

tp_ev_pair.dnrec2: the difference in the number of reconstructed hits between the longest and third longest segments associated with the Monte Carlo track.

tp_ev_pair.drow1: the difference in the row numbers of the closest endpoints of the two longest segments associated with the Monte Carlo track.

tp_ev_pair.drow2: the difference in the numbers of the closest endpoints of the longest and third longest segments associated with the Monte Carlo track.

tp_ev_pair.ddist1: the distance, in cm, between the closest endpoints of the two longest segments associated with the Monte Carlo track.

tp_ev_pair.ddist2: the distance, in cm, between the closest endpoints of the longest and third longest segments associated with the Monte Carlo track.

tp_ev_pair.mcptg: the generated transverse momentum of the Monte Carlo track.

tp_ev_pair.mcpzr: the generated longitudinal momentum of the Monte Carlo track

tp_ev_pair.acent(4): the x-center of the helix projection onto the x-y plane of each segment listed in order of decreasing nrec.

tp_ev_pair.bcent(4): the y-center of the helix projection onto the x-y plane of each segment listed in order of decreasing nrec.

tp_ev_pair.rad(4): the radius of the helix projection onto the x-y plane of each segment listed in order of decreasing nrec.

tp_ev_pair.phi0(4): the phi0 helix parameter of each segment listed in order of decreasing nrec.

tp_ev_pair.tanl(4): the dip angle helix parameter of each segment listed in order of decreasing nrec.

The tp_mctrk table contains one entry per Monte Carlo track that enters the TPC and provides the following set of information about that track:

tp_mctrk.mcid: the Monte Carlo track identifier. This is the primary key for the mkine table.

tp_mctrk.nfst: the number of points on the track generated by the tfs module.

tp_mctrk.ngnt: the number of points on the track generated by GEANT.

tp_mctrk.nrec1: the number of points assigned to the longest reconstructed segment by the tracking program.

tp_mctrk.nrec2: the number of points assigned to the next to the longest reconstructed segment by the tracking program.

tp_mctrk.nseg: the number of segments reconstructed from this Monte Carlo track by the tracking program.

tp_mctrk.pid: the particle ID for this track as assigned by GEANT.

tp_mctrk.recid: the id of the longest reconstructed segment. This is the primary key of that segment's entry in the tptrack table.

tp_mctrk.vid: the vertex id for the Monte Carlo track as extracted from the GEANT information.

tp_mctrk.ptg: the transverse momentum at the first point of the Monte Carlo track.

tp_mctrk.ptr: the reconstructed transverse momentum at the first point of the track.

tp_mctrk.pzg: the longitudinal momentum at the first point of the Monte Carlo track.

tp_mctrk.pzr: the reconstructed longitudinal momentum at the first point of the track.

tp_mctrk.qfact: the quality factor for this track defines as the number of good points assigned to this track divided by the total number of points on the track.

Section 4.3. TPC Track Evaluation Usage

This section will briefly describe how to incorporate the tte module in a TAS executable. The standard tte.mid (Module Interface Definition) file must be pointed to when constructing the TAS executable. This .mid file is as follows:

```
table_lib mct
table_lib tcl
table_lib tpt
table_lib tte
#
module tte tte_main
#
module_table    tte    tptpar      read     1
module_table    tte    tptrack     read     2
module_table    tte    tphit       read     3
module_table    tte    mhits_tpc   read     4
module_table    tte    mkine       read     5
module_table    tte    mpart       read     6
module_table    tte    mvert       read     7
module_table    tte    tpeval      write    8
module_table    tte    tp_mctrk    write    9
module_table    tte    trkres      write    10
module_table    tte    tpeval_fit  write    11
module_table    tte    tp_ev_pair  write    12
module_table    tte    ev_cntrl    write    13
```

Other than that, the only other latitude given to the tte user is through the ev_cntrl table described in the previous section. If problems raise their ugly head, make sure that the following tables are filled: tptpar, tptrack, tphit, mhits_tpc, mkine, mpart, and mvert. The minimum STAR analysis chain that must be run for proper tte operation is the following: mct -> tfs -> tpt -> tte. Happy evaluating!

BIBLIOGRAPHY

1. J. Knoblock and P. Norton, The Status of Reconstruction Algorithms for ALEPH.

2. See the documentation in the mct section of the STAR World Wide Web.

3. P. Jones, STAR Note 56.

4. M. Hansroul, et. al., Nucl. Inst. Meth. **A270** (88) 498, for example.

5. N. I. Chernov and G. A. Oskov, Effective Algorithms for Circle Fitting, Comp. Phys. Comm. **33** (94) 329.

6. R.K. Bock, et. al., Data Analysis Techniques for High Energy Physics Experiments, Cambridge University Press, 1990.