E-949 K-064
November 17, 2006

Experiment 949
Technical Note K-064

Target Reconstruction Appendix. A guide to TG coding.

Benji Lewis

Abstract

This appendix is meant to be a guide or manual to the subroutines which are employed by *TGrecon*. The description that follows is by no means complete. This is indended to point a programmer in the right direction. Many of the routines (especially in tgrecon_commands.F) are simple and are straight forward to read the routine directly. However, some are very complicated and to obtain further understanding a user must read all comments and have an understanding of the target geometry. Much of the array structure had to be carried over from *swathccd* for compatibility with the rest of the KOFIA and PASS2 programs.

All of the subroutines that make up *TGrecon*are placed in files in the following format TGrecon_xxx.F. Where xxx are *commands*, *geometry*, *cluster*, *matching*, *routines*, and *fill*. The following sections the subroutines are discussed.

The variable names are meant to be descriptive. More complicated routines have comments within the program to aid in understanding.

A *swathccd's* common block *swathccd.cmn*

These variables are used by *swathccd* , so understanding these are important in understanding the how *TGrecon*is implemented.

- iqual_swccd - (integer) - quality of the *swathccd* reconstrucion.

  =0, good fit and has a pion track

  =1, good fit without a pion track, but kaon is at edge of target.

  =2, no pion track along the swath.

  =3, no kaon cluster connected to the swath.

  =4, an error occurred, see ierr_swccd

- ierr_swccd - (integer) -

  =0, no error

  =1, unable to find hits.

  =2, number of cluster hits were exceeded.

  =3, more than 5 kaon clusters found.

- nclustccd - total number of clusters in *clustccd*.

- nmembccd(i) - number of hits in cluster i. Maximum $i = 5$
- clustccd(3,j,i) list of kaon clusters

  clustccd(1,j,i) = column # in pulse j of cluster # i.

  clustccd(2,j,i) = row # in pulse j of cluster # i.

  clustccd(3,j,i) = hit # in pulse j of cluster # i.
- swnclustccd - total number of kaon-like clusters connected to the swath (SW clusters).
- swclstagccd(5) - pointers to those clusters that are connected to the swath, points to the i^{th} cluster in *clustccd*.
- swkpntrccd - the index in *swclstagccd* pointing to the best kaon cluster.
- Cwccd(3,j,i) - Same as clustccd. This is the pion supercluster.
- icwccd(i) - The total number of members in cluster *Cwccd(,,i)*.
- CwEnccd(j,i) - Energy of the pion cluster hit from *Cwccd(,j,i)*.
- CwTimccd(j,i) - Time of the pion cluster hit from *Cwccd(,j,i)*.
- CwLkccd(j,i) - Output of the likelihood function of the pion cluster hits from *Cwccd(,,i)*.
- CCWccd(3,j,i) - Couter-Clockwise pion cluster.
- ICCWccd(i) - total # of hits from cluster *CCWccd(,,i)* on the swath along the opposite direction of the pion track from *Cwccd*.
- CCWENccd(j,i) - Energy of the cluster hit from *CCWccd(,j,i)*.
- CCWTIMccd(j,i) - Time of the cluster hit from *CCWccd(,j,i)*.
- SWccd(3,j,i) - Same as *Clustccd* except that are kaons that connect with the swath.
- ISWccd(i) - Same as *nmembccd* except that are kaons that connect with the swath.
- SWENccd(j,i) - Energy of the cluster hit from *SWccd(,j,i)*.
- SWTIMccd(j,i) - Time of the cluster hit from *SWccd(,j,i)*.
- NGAMMAccd - Total # of photon hits
- GAMMAccd(k,j) - List of photon hits in the target.

  k=1, column # in hit # j

  k=2, row # in hit # j

  k=3, hit # in hit # j
- GAMENccd(j) - Energy of hit j from list *GAMMAccd(,j)*.
- GAMTIMccd(j) - Time of hit j from list *GAMMAccd(,j)*.

- SWMINSPRccd(i) - Closest distance to the DC track among the hits in *SWccd(,,i)* SW-cluster (the corresponding fiber is referred to as the "nearest member").
- KPIMINccd(i) - Minimum distance between the "nearest member" and any of the hits in cluster *CWccd(,,i)*
- KPIMAXccd(i) - Maximum distance between the "nearest member" and any of the hits in cluster *CWccd(,,i)*
- TIMEK_SWccd - Average kaon time hits from cluster *SWccd(,,swkpntrccd)*.
- TIMEKS_SWccd - Standard deviation of kaon time hits from cluster *SWccd(,,swkpntrccd)*.
- TIMEP_SWccd - Average pion time hits from cluster *CWccd(,,swkpntrccd)*.
- TIMEPS_SWccd - Standard deviation of pion time hits from cluster *CWccd(,,swkpntrccd)*.
- NTRIK_SWccd - # of kaon hits
- NTRIP_SWccd - # of pion hits
- Like_SWccd - # combined likelihood of pion fibers.
  i.e. $\sum_j(CwLkccd(j, swkpntrccd))$
- Like2_SWccd - Same as *Like_SWccd* except the likelihood function uses only distance.
- Xin_SWccd - X position of B4 hit.
- Yin_SWccd - Y position of B4 hit.
- DB4_SWccd - Distance between B4 and nearest fiber.
- DB4TIP_SWccd - Distance between B4 and nearest extreme tip of K cluster.
- DVXTIP_SWccd - Distance between found decay vertex and nearest tip.
- QB4_SWccd - TRUE if B4 info was used on this event. More specificly, it is TRUE, if the input values *(xin,yin)* $\geq$ 8.0cm.
- PERIFX_SWccd - x of pion at the target edge (in target coordinates)
- PERIFY_SWccd - y of pion at the target edge (in target coordinates)
- IEDGE_SWccd - edge # (1 to 6) at which the pion left the target (consistent with the I-counter numbering)
- XSTPKccd(2) - x for two extreme possible Kaon stopping points
- YSTPKccd(2) - y for two extreme possible Kaon stopping points
- STX_SWccd - average x of Kaon stopping point
- STY_SWccd - average y of Kaon stopping point
- STZ_SWccd - z of Kaon stopping point

- ENERK_SWccd - energy of Kaon in target
- ENERP_SWccd - energy of Pion in target
- ENERG_SWccd - energy of photon in target
- PLNTH_SWccd - pathlength of the pion in x-y plane
- DPLNTH_SWccd - error in the pathlength
- PLNTHV_SWccd - pathlength of the visible pion track in x-y plane
- DVXPI_SWccd - Distance from supposed K decay vertex to closest pi fiber
- KPIANG_SWccd - Angle between K and pi
- DEXTIP_SWccd - Distance between extreme tips of K cluster
- DTARGF_SWccd(5) - Real nearest K-Pi triangle separation, analogous to the TN186 TARGF routine. Note that the above KPIMIN is essentially meaningless, and should be ignored.
- NEV_SWccd - Event number for which this data is valid.
- NRN_SWccd - -Run number for which this data is valid.
- NTRK_SWccd - DC track number for which this data is valid.

B Commonly used variable names

Most indices are associated in the following way:

- i = cluster #
- ii = points to a different i
- j = member # of a given cluster
- k = 1 - column, 2 - row, 3 - hit #
- *nc,nc1,nc2* usually refers to a specific cluster #
- *nclust* - the total number of clusters available
- *col* = TG fiber column number
- *row* = TG fiber row number
- *hit* = hit number within a particular fiber (there can be a maximum of 2 hits in each fiber).
- *Cand(3,413) or List(3,413)*
  Cand/List(1,i) = column of pulse
  Cand/List(2,i) = row of pulse
  Cand/List(3,i) = pulse #

- *Clust(1,j,i)* = column in element j of cluster i
- *Clust(2,j,i)* = row in element j of cluster i
- *Clust(3,j,i)* = pulse # in element j of cluster i

C TGrecon_commands.F

TGrecon_commands are a set of functions and subroutines that do basic action. Very much like copy and remove commands in operating systems. I have tried to place error catching statements when a routine tries to exceed sizes of arrays or attempts to use invalid array indices.

C.1 duplicate_cand(Cand,ncand,col,row,hit)

This is a logical function that checks to see if this pulse/hit is already in the candidate list, *Cand*. Returns T if the same hit is already within the list. Returns a F otherwise.

C.2 duplicate_clust(Clust,nMemb,nc,col,row,hit)

This is a logical function that checks to see if this pulse/hit is already in the cluster number *nc* from supercluster *Clust*. Returns T if the same hit is already within the cluster. Returns a F otherwise

C.3 copy_cand(List1,nlist1,List2,nlist2)

Copies the *List1* (with *nlist1* elements) into a new list, *List2*

C.4 copy_clust(Clust1,nMemb1,nclust1,Clust2,nMemb2,nclust2,maxclust2,omit_zeros)

Copies supercluster *Clust1* into a new supercluster named *Clust2*. If *omit_zeros* is F, then the *Clust2* is identical to *Clust1*. If *omit_zeros* is T, then any empty clusters within *Clust1* are skipped during the copying.

C.5 copy_clust_pi(...)

This routine is the same as the *copy_clust* routine except that it defines *Clust2* as `Clust2(3200,*)`, instead of the standard cluster array definition of `Clust2(3150,*)`,. This routine had to be created since in *swathccd* the kaon and pion clusters have a different maximum number of elements allowed in each cluste (150 for kaons and 200 for pions). In *TGrecon*, all clusters have a maximum number of 150. This allows for more generic routines.

C.6 Remove_elem_Cand(List,nlist,rm_el,quick)

As the name implies, this removes element number *rm_el* from *List* (which has *nlist* total elements). If *quick* is T, then ordering of the list is not preserved. When *quick* is T the routine is faster. If *quick* is F, then ordering of the list is preserve (which slows down the routine).

C.7 Remove_elem_Clust(Clust,nMemb,nc,rm_el,quick)

As the name implies, this removes element number *rm_el* from cluster number *nc* from the supercluster *Clust* (which has *nMemb(nc)* total elements). If *quick* is T, then ordering of the cluster is not preserved. When *quick* is T the routine is faster. If *quick* is F, then ordering of the cluster is preserve (which slows down the routine).

C.8 Add_elem_Cand(Cand,ncand,col,row,hit)

Adds an hit to the candidate list *Cand*. The hit number *hit* from fiber with row number *row* and column number *col*. Note that *hit* is 1_st, 2_nd, 3_rd, etc. hit in a given fiber. The hit is added to the end of the list.

C.9 Add_elem_Clust(List,elem,Clust,nMemb,nc)

The hit *elem* from the *List* is added to the cluster *Clust(,,nc)*. The hit is added to the end of the list.

C.10 Clust_Order(Clust,nMemb,from,to)

Changes the cluster order around. Does not change the order of individual cluster list. Moves cluster *from* $\leftrightarrow$ *to*. That is if we have a supercluster A with 3 clusters (a,b,c) and we did the following
`Clust_Order(`A,$nMemb\_A$,1,3)
Then the clusters would be order in the following way *c,b,a*.

C.11 Clust_Memb_Order(Clust,nMemb,nc,from,to)

Changes the member order of the specific cluster *Clust(,,nc)*. Very similiar to *Clust_Order* except the individual elements of a given cluster will be changing order. This routine is used in *Track_Order* in which (most cases) attempts to order the pion track from decay point to exit point.

C.12 remove_zero_clust(Clust,nMemb,nclust)

It is possible via the *Remove_elem_Clust* to remove all element from a given cluster. Since an empty cluster is of no use to the programmer and possibly confuse other routines *remove_zero_clust* was created. It's purpose is to scan through the supercluster *Clust* and remove null clusters from the cluster list.

C.13 Double_Hit(Clust1,nMemb1,nc1,Clust2,nMemb2,nc2,double,ndouble)

From the two input clusters, *Clust1(,,nc1)* and *Clust2(,,nc2)*, this routine will return a list, *double(2,ndouble)*, of the fibers that are in common to both clusters.
double(1,i) points to the member index number from *Clust1(,,nc1)*
double(2,i) points to the member index number from *Clust2(,,nc2)*

C.14 Repeat_Hits(Clust1,nMemb1,nc1,Clust2,nMemb2,nc2,double,ndouble)

This is identical to *Double_Hit* except that it looks for a hit in common (not a fiber). Remember that a fiber can have multiple hits.

C.15 Order_Cluster(Clust,nMemb,nc,Order)

This changes the order of the members of *Clust(,,nc)* given by an ordering array as defined by the routine *Track_Order*. Uses *Clust_Memb_Order* to accomplish the task.

C.16 test_change_utc

A temporary routine to rotate the UTC track about a point. Use to help study *Kink_Finder*.
This routine is of no use after the study is complete.

C.17 rotate_utc

A more general version of *test_change_utc*.
This routine is of no use after the study is complete.

D TGrecon_geometry.F

TGrecon_geometryis a set of routines that deal with the geometry within the E949 target so that reconstruction of the target is possible or at least made easier. Although most routines in the *TGrecon*environment uses geometry to so degree, these routines represent the basics.

D.1 arc_length(r,x1,y1,x2,y2)

A function that returns the length of an arc on a circle of radius *r* from the points (*x1,y1*) to (*x2,y2*). The calculation has two solutions. The smallest is the returned value.

D.2 translate(x,y,dx,dy)

Translates a point (*x,y*) to (*x+dx,y+dy*)

D.3 rotate(x,y,theta)

Rotates point *(x,y)* about the orgin by angle *theta*.

D.4 circle_inter(xc1,yc1,r1,xc2,yc2,r2,x,y)

Given two circles with radius' and center point as *r1, (xc1,yc1) and r2, (xc2,yc2)* this routine will return the points,*(x(1),y(1)),(x(2),y(2))* where the circles intersect. If the circles do not intersect the returned values will be (999.,999.).

D.5 lines_inter(m1,b1,m2,b2,x,y)

Returns the intersection point *(x,y)* of two lines with slopes and y-interscepts of *m1,b1 and m2,b2*.

D.6 lines_angle(m1,b1,m2,b2,tan_ang)

Returns the angle *tan_ang* between two lines with slopes and y-interscepts of *m1,b1 and m2,b2*.

D.7 get_neigh(iel,iro,NbrList,nnbrs)

Does the same as *Nbrs* (subroutine within *swath.F*), but removed the edge fiber stuff (this is possibly a mistake to do, need to look futher).
Input a fiber column number *iel* and row number *iro* and *get_neigh* will return a list of fibers that neighbor the input fiber.
nbrlist(1,) = column number of neighboring fiber*
nbrlist(2,) = row number of neighboring fiber*
nnbrs = total number of neighbors

D.8 find_neighbors(icol,irow,Neighbors,nneigh)

This is an expanded version of *get_neigh* so that gaps are allowed.
Input a fiber (*col,row*) and this routine will return an array with the adjacent and gap neighbors, see Figure 1.
Neigbhors(1,1,index) = adjacent neighbors column number
Neigbhors(1,2,index) = adjacent neighbors row number
Neigbhors(2,1,index) = gap neighbors column number
Neigbhors(2,2,index) = gap neighbors row number
nneigh(1) = total number of adjacent neighbors
nneigh(2) = total number of gap neighbors

D.9 find_clust_neigh(Clust,nMemb,nc,Neighbors,nneigh)

Given a cluster,*Clust(,,nc)* ,this routine will find all fibers that are neighboring the cluster. NOTE: The routine is currently not working for gap neighbors. The arrays *Neighbors* and *nneigh* are in the same format as in *find_neighbors*.

D.10 position_to_fiber(xin,yin,row,col,nfib)

Given an (*x,y*) point in space this routine will return a fiber (*row,col*) and overall fiber number, *nfib*, which is calculated by $fiber\# = column\# + 24 \times row\#$.

D.11 track_to_fiber(x0,y0,radius,dist_to_fiber)

This routines returns an array, *dist_to_fiber(24,23)*, whose element values are the distances from each fiber to the closest approach from a track with *radius* and center point of (*x0,y0*).
dist_to_fiber(col,row) = distance from track to fiber (*col,row*).

D.12 min_max_dist_clust(Clust,nMemb,nclust,do_track,x,y,mn_pnt,mn_dis,mx_pnt,mx_dis)

Searches all clusters within the supercluster, *Clust*, and finds the fiber (within each cluster) that is the closest to and futherest away from either a point (*x,y*) or a UTC track. If *do_track* is T, then the routine uses for distance the array Spr(col,row) (within tgrecon.cmn) as the track

distances to the fibers. If *do_track* is F, then the routine uses the distance from the point (*x,y*) to the fiber in question within the cluster. The returned variables are as follows: Input Values: *Clust,nMemb,nclust,do_track,x,y*
Returned Values: *mn_dis(i)* = the minimum distance from cluster i to the point (*x,y*) or Spr. *mn_pnt(i)* = the index value of the minimum-distance fiber from cluster i to the point (*x,y*) or Spr. *mx_dis(i)* = the maximum distance from cluster i to the point (*x,y*) or Spr. *mx_pnt(i)* = the index value of the maximum-distance fiber from cluster i to the point (*x,y*) or Spr.

NOTE: This routine is dependent upon the array *Spr* within *tgrecon.cmn* to be filled first (if *do_track* is T).

D.13 Neighboring_Clust(...)

Input Values: *Clust1,nMemb1,nclust1,Clust2,nMemb2,nclust2, board_thres*
Return Values: *mn_dist,mx_dist,mn_dist_pnt,mx_dist_pnt,Border,nCBorder,nborder*

- *board_thres* is the threshold of what is considered to be a bordering neighbor.
- *mn_dist(i,j)* = the minimum distance from *Clust1(,,i)* to *Clust2(,,j)*.
- *mx_dist(i,j)* = the maximum distance from *Clust1(,,i)* to *Clust2(,,j)*.
- *mn_dist_pnt(k,i,j)* , *mx_dist_pnt(k,i,j)* are index of the fibers that are min and max distances from each other in *Clust1(,,i)* to *Clust2(,,j)*.

  when k=1, then this is the pointer to the element of from *Clust1(,,i)*

  when k=2, then this is the pointer to the element of from *Clust2(,,i)*
- *Border(i,j)* = Does 0 *Clust(,,i)* border *Clust(,,j)* If no border; 1 if there is a adjacent border; 2 if there is a gap border.
- *nCBorder(i)* = the total number of clusters that border *Clust(,,i)*
- *nborder* = total number of clusters that border.

D.14 Closest_Cluster(Clust,nMemb,nclust,xin,yin,clust_dist,closest_clust)

Finds the closest cluster from *Clust* to the point (*xin,yin*). The return values are the cluster number that is the closest *closest_clust* and the distance *clust_dist*. This routine calls *min_max_dist_Clust*. This routine is helpful to determine the best kaon cluster relative to the B4 position of the Kaon.

D.15 TG_Edge(Clust,nMemb,nclust,edge,nedge)

Searches all clusters within the *Clust* to determine if the cluster borders the edge of the target. *edge(i)* will be 1 if cluster *i* is bording the edge (0 otherwise). *nedge* is equal to the total number of clusters that do border the edge.

D.16 End_Points(Clust,nMemb,nclust,endpt,endpt_dist)

Finds the end points of all clusters in *Clust* by first determining the hit furthest away from the center of the target, point (0,0), then finding the hit that is furthest away from the first fiber. Indices to the fibers, from cluster *i*, are in *endpt(1,i)* and *endpt(2,i)*. *endpt_dist(i)* is the distance from one end point to another in cluster *i*.
Note that *Extreme_Tips* and this routine are very similiar, but *Extreme_Tips* seems to be alittle more robust.

D.17 Extreme_Tips(Clust,nMemb,nclust,endpt,endpt_dist)

Finds the extreme tips of all clusters in *Clust* by determining the two fibers that are furtherest away from one another. *endpt(1,i)* and *endpt(2,i)* are indices to the extreme tips of cluster *i*. *endpt_dist(i)* is the distance from one tip to another from cluster *i*.

Note that spurs (i.e. other unrelated tracks that unexpectitly joins the real track) could cause a wrong tip to be picked. Possible improvements to this could include some type of ordering of the fibers from expected tips.

E TGrecon_cluster.F

Routines dealing with clustering of target hits are discussed within this section. Clustering is defined as grouping hits together that are in close geometrically.

E.1 Cluster_Cand(List,nlist,Clust,nMemb,nclust,maxclust,do_gap,ierr)

Given list of candidate hits, *List* with *nlist* hits, (usually created by *Classify_Cand*), creates a supercluster, *Clust*. The maximum number of clusters allowed in *Clust* is *maxclust*. If *do_gap* is T, then the clustering allows the clusters to have gaps seperating the fibers, see Figure in TGrecon technote. If *do_gap* is F, then when one looks at the fibers from a cluster the cluster will be a solid object, see same figure.
This routine is modeled after a biological virsus infecting other neighboring cells. One cell (fiber/hit) is infected (with the virus) and then the newly infected cell passes the virsus along as well. Once all of the environment (all neighboring hits) are infected, a search through the list for uninfected (not assigned to a cluster yet) hits is done. When a new hit, within *List*, is found a new virus is 'seeded' to this hit and then we let the infection spread. Each new cluster of hits is analogous to a different strain of the virus.

E.1.1 completion(List,nlist)

A logical function the is used within *Cluster_Cand* to determine if all the hits have been assigned to a cluster.

E.1.2 infect(List,nlist,elem,group,do_gap)

A routine made for *Cluster_Cand* to place (or infect) neighbors of the current cluster in question, *group* is the current cluster number.

E.2 Mend_Fibers(...)

Input: (Clust,nMemb,nclust,origList,orignlist,Twindow,QTIME,do_gap,do_grow,do_rem,QMEND)

There is a need to add new hits to a preexisting cluster. *Mend_Fibers* is constructed in the most general way as possible. You give this routine a existing supercluster *Clust* and a list of hits, *origList*, that you would like the algorithm a chance of mending into the clusters. Note that *Mend_Fibers* is not forced to use all the hits from *origList*. The average time of the original clusters are calculated and a time window is created by the input variable *Twindow*. The time window is $(avgtime_i - Twindow) \leq hittime \leq (avgtime_i + Twindow)$, such that i is the cluster number.

Logical Switches:

- *qtime*: If true, will impose the time window on the possible list of hits to mend. If false, no time window is imposed on the list of hits to mend.
- *do_gap*: If true, mends hits that are adjacent neighbors and are gap neighbors to the cluster. If false, only mends hits that are adjacent to the cluster.
- *do_grow*: If false, only considers hits possible for mending that are neighboring (what a neighbor is is defined in *do_gap*) the original cluster. If true, allows mending of neighbors of new mended hits as well as the original cluster. As the name implies if it is true, the cluster can grow much more readily. With this in mind, the programmer should be careful when selecting this switch as true.
- *do_rem*: If false, *origList* remains the same. If true, any hits that are mended into a cluster in *Clust* are removed from *origList*.

E.3 Combine_Clusters(Clust,nMemb,nclust,Clust2,nMemb2,nclust2,maxclust,do_gap)

Given two superclusters, *Clust, Clust2*, the routine submits the list to the *Cluster_Cand* program for clustering. This routine omits hits that occur in both superclusters.

E.4 Re_Cluster(Clust,nMemb,nclust,maxclust,do_gap)

On some occasions such as when hits are mended to or removed from the clusters within *Clust* individual clusters within the supercluster combine together to form a larger cluster. Or possible a cluster breaks apart into two or more pieces when hits are removed. What *Re_Cluster* does is to submit all the hits from the supercluster to *Cluster_Cand*.

F TGrecon_matching.F

The matching division deals with finding the best cluster-to-cluster match between the pion clusters and all of the kaon clusters (kaon and kaonlate). Cluster matching is discussed in some detail in the clustering section of this technote. These routines are a broader verision that were inbedded within *TGrecon*, but grew in size and complexity. So for clarity, I broke the matching into a seperate division. However, these routines are not created to be very general in nature (partly due to their beginnings). They use variables stored in the tg_matching common block in tgrecon.cmn.

All 4 routines described in this section have the following inputs and one output value:

- *xin,yin* the value in which you believe the Kaon entered the target. i.e. B4 entering point. If the values are ≥ 8.0 then the value are assumed to be unknown, the target has a radius of only 6cm.
- *xexit,yexit* the value where you believe the pion exited the target.
- *matching* a flag value returned by the routine to state the cluster matching results.
  = 5, A match found between a kaon cluster and a pion cluster that is on the swath.
  = 7, A match found between a kaon cluster and a pion cluster that is not on the swath.
  = 13, Error occurred during an attempt to match a swath pion cluster to a kaon cluster.
  = 15, No Pion/Kaon matching. This is a very basic failure.

F.1 Match_Clusters

This is the main matching routine, the subroutines that follow are called from here. The superclusters in question at this point are *PClust*, *KClust*, and *KlateClust* (pion cluster, kaon and late kaon cluster). The matching tries to find a pion cluster that posses 'matching' qualtities to either a kaon or late kaon cluster.
The overall problem with matching is what criteria is most important. The problem becomes worse when the importance of the criteria changes depending on what type of cluster match you are dealing with. One of *Match_Clusters*'s purpose is to find what match category you are working with.

Initial Matching Criteria

- The average time of a pion cluster must be greater than the average time of the kaon cluster. I have a hard coded change for this criteria. Since the pion fiber time is somewhat uncertain I have placed in a $1ns$ buffer. That is, a match is possible only if $avgtime_{pi} \geq avgtime_K - 1$.
- The pion cluster and kaon/kaonlate cluster must be neighboring.

At this point, the kaon and kaonlate clusters carry the same weight. This was not the case in earlier version of matching and is not the case in *swathccd* . In *swathccd* if there is anyway to get a kaon cluster to match it is accepted, even if a kaonlate cluster is a better match.

Secondary Matching Criteria

- Pion cluster on the swath. Having pion clusters on the swath is important for any event with a valid UTC track, which includes all triggers except KBeam.
- Having any match.

The matching is highly dependent upon the *Neighboring_Clust*. Initial swath, *swidth*, is 1.0. Initial neighbor distance 1.2. Since having a pion cluster on the swath is currently the most important criteria to statisfy, *swidth* is increased to a maximum of *swidth_thres* (2.2). If a swath-pion match is not yet found, we increase the range of what a neighbor is considered to be, 2.0cm. If at this point a pion-swath match is not found, we have to give up the hope of having a pion cluster on the swath that has a adjoining kaon cluster. If there is a match then a call to *Match_SwP_K* is made to finishing the matching.
If no swath-pion match is made, then we default to any match. And a call to *Match_PK* is made.

F.2 Match_SwP_K

This routine is called by *Match_Clusters* when we have pion clusters that are on the swath. Pion clusters on the swath are the ideal candidates to reconstruct events in which PNN1/PNN2 are interested in analyzing.

F.3 Match_PK

This routine is an attempt to reconstruct a kaon decay within the target. This routine is called when we do not have pion clusters that on located on the swath.

F.4 Match_edgeK

This was an attempt to reconstruct events when the Kaon is located on the edge of the TG. In such cases, no pion hits are observed. This routine is not called by *Match_Clusters* and is *Match_Clusters* not currently stable.

G TGrecon_routines.F

This division is a set of routines that have more of a specific purpose and can not fit within different class of subroutines.

G.1 Classify_Cand

Creates a list of candiates with windows in time and energy. When *dowidth* is true, the time of the hit is considered the sum of fib_t and fib_w. The *dowidth* switch is in place due to *swathccd* having a candidate list that is created this way. *TGrecon*has *dowidth* always false.

G.2 Remove_Double_Hits(Clust1,nMemb1,nc1,Clust2,nMemb2,nc2,removed1,removed2)

Give this routine two clusters, *Clust1(,,nc1), Clust2(,,nc2)* and it will remove the hits that are in both clusters. The logical varibles *removed1* or *removed2* will be true when a hit is removed from *Clust1(,,nc1)* or *Clust2(,,nc2)*, respectively. Any hit that is in both clusters will remain in the cluster whose average time the hit is closest.

G.3 Omit_Outliers(Clust,nMemb,nclust,Clust2,nMemb2,nclust2,twindow,do_dweight)

From a supercluster, *Clust*, generates a new supercluster, *Clust2*, that has omitted outliers in time. The purpose of this routine is to remove hits that do not belong in the cluster. The is determined by calculating the average time value of each cluster and testing whether ... If *do_weight* is true a weight of $e^{-|x|}$ (x is the d.c.a distance of the hit fiber and the UTC track) and is applied to the time of the hit.

NOTE: the UTC track is defined through *tgrecon.cmn* by the varible Spr. Hence Spr should be filled before calling this routine if *do_weight* is true.

G.4 likelyhood(dtime,do_t,energy,do_e,dist,do_d)

This function is copied from *swathccd* .F likelihood function (they are spelled differently note the y and i). *swathccd* version is dependent on the variables *dtime* (time difference), *energy*, and *dist* (d.c.a. to the UTC track). This version will turn off the depencence of any of these variables by making one of the following switches false, *do_t, do_e, do_d*. This could be useful when the UTC track is not realiable such as in TG scattered events.

G.5 Like_Omit(Clust,nMemb,nclust,avg_time,like_thres,do_t,do_e,do_d,Cand,ncand)

do_t, do_e, do_d are the switches for the likelyhood function used within this routine. *avg_time* is the average times that the programmer would like to use for clusters in *Clust* (this could be the average time after omitting outliers, etc.). If the likelyhood function returns a value less than *like_thres* the hit is removed from the cluster. All hits removed are placed in the hit list *Cand* for later use.

G.6 Like_Omit_Pnt

Basically a copy of *Like_Omit* except: (1) Looks only at one cluster. (2) Does no remove hits from the cluster. *List* points to the index values of all hits that does not pass the likelyhood threshold. (3) Track information is not assumed to be Spr. User must input the radius and center of circle *r, x0, y0*.

G.7 Cluster_on_Swath(swidth,Clust,nMemb,nclust,SwClust,n_onSwClust,nswclust)

Uses the *Spr* array to determine the distance from the swath to the fiber. The width of the swath is set by *swidth*. *SwClust* points to the clusters that have hits on the swath. *n_onSwClust(i)* indicates the number of elements from cluster *i* on the swath. *nswclust* the total number of clusters from *Clust* that are on the swath.

G.8 Avg_time_Cluster(Clust,nMemb,nclust,avg_time,std_time)

Returns the average time and standard deviation in a cluster. The average time is -999. if no members exist and standard deviation is 999. if less than two members exist.

G.9 Avg_time_Cluster_weight

Similar to *Avg_time_Cluster* with the following exceptions: Note that when *do_window* = F and *weight_type* = 0, *Avg_time_Cluster* returns the same value as *Avg_time_Cluster_weight*

- *weight_type*
  - = 0, applies a weight of 1.
  - = 1, applies a weight from *sw_weight_ccd*.
  - = 2, applies a weight from *sw_weightk_ccd*.
- *do_window*
  - = T, do not include hits outside the time window.
  - = F, there is no time window

- *time_low,time_high* defines the time window.

G.10 Clust_to_Clust_tracking

This routine tracks the movement from *Clust1(,,nc1)* to *Clust2(,,nc2)*. Assumes the starting point of the particle is in *Clust1(,,nc1)* at point (*xin,yin*) then the track ends and *Clust2(,,nc2)* tracks begins. Created to help find the decay vertex of the Kaon. The routine returns the point (*vertex_x, vertex_y*) (i.e. the decay vertex) which is the end point of the *Clust1(,,nc1)*'s track and the beginning of *Clust2(,,nc2)*'s track. The ending point of *Clust2(,,nc2)*'s track is the point (Ext_pnt_x1,Ext_pnt_y1).

G.11 Track_Order(Clust,nMemb,nc,xin,yin)

This routine orders the hits within *Clust,,nc* so that the first hit is fiber at or nearest to the point (*xin,yin*). Modeled similiar to the biological infection scerino used in *Cluster_Cand*.

G.11.1 trk_complete(Order,norder)

Logical function used by *Track_Order* to determine if the *Track_Order* has finished tracking.

G.11.2 seed_order(Clust,nMemb,nc,Order,ord,f_Order,f_ord)

A routine used by *Track_Order* to find the next hit to seed (with an ordered infection) when *Track_Order* was unable to find the next closest neighbor.

G.11.3 infect_order(Clust,nMemb,nc,memb,Order,ord,f_Order,f_ord,do_gap,found)

A routine used by *Track_Order* to find and infect the next hit with next order number.

H TGrecon_fill.F

This set of routines is what makes *TGrecon*fairly transparent when used instead of *swathccd* . This routine fills variables from *swathccd* 's common block, *swathccd.cmn*, with variables from *TGrecon*'s common block, *tgrecon.cmn*.

H.1 Initialize_Common

This routine initializes all values from all common blocks directly associated with the reconstruction,*swathccd.cmn* and *tgrecon.cmn*. Most values are set to zero, 999., or -999. Making the values in question out-of-range, so that if these values are not assigned anything by a subroutine then it will stay beyond the true value region.

H.2 Last_Common

This routine fills *swathccd.cmn* values that are not need until *TGrecon*is finished.

H.3 Fill_Gamma

Fills variables associated with the Gamma particle, in the copied from a block of code from *swathccd*.

H.4 Fill_Aux_Arrays

This is more of debugging routine than anything else. What it does is places all of the hits that were part of the pi, kaon candidate lists and any hit that was not assigned to the final pion and kaon clusters (and the gamma hits) are assigned to new auxiliary arrays placed from a new common block. These arrays could be viewed in PAW Photo and determine if the Algorithms are picking the wrong Kaons and Pions. This routine should be disabled after all studies are done to *TGrecon*.

H.5 Energy_Time(*tpiext*)

This routine similiar to Fill_Gamma and Last_Common is to fill *swathccd.cmn* variables in the same way *swathccd* would fill the variables. This block of code is identical to code from *swathccd* .

I Kink_Finder routines

I.1 Cand_linefit2(x1,y1,List,nlist,err_x,err_y,dca_min,m,b,rsq,sigma_m,error)

The routine which is currently used. *x1,y1* are the arrays which store the (x,y) coordinate values. *List* is a pointer array which indicate the indices that you would like to use from the x,y arrays. *nlist* is the number in *LIst*. *dca_min* has been disabled. The rest of the input and outputs are passed to least_sq_linefit().

Currently, this routine is called by *KinkFinder* and sends the distance along the UTC track (starting) from the exit point of the target as the *x1* array and *y1* is the distance of closest approach to the fibers in the cluster of interest within *KinkFinder* .

I.2 Cand_linefit(Clust,nMemb,nc,List,nlist,err_x,m,b,rsq,sigma_m,error)

Similar to Cand_linefit2 except it uses the (x,y) coordinates of a cluster hits.

I.3 least_sq_linefit(x,y,elements,err_x,err_y,exclude,m,b,rsq,sigma_m,error)

- *x,y*: list of x and y coordinates to perform the fit on.
- *elements*: total number of elements in x and y
- *err_x,err_y*: not used.
- *exclude* logical array which if $exclude(i)$ is true will exclude the point $x(i), y(i)$ from the fit.
- *m,b,rsq*: slope, y-intercept, and R^2 (correlation) of the fit
- *sigma_m*: standard deviation of m (slope)
- *error*: error flag, informs user what caused an error

I.4 get_kink_list_oldway(Clust,nMemb,nc,dca_min,List,nlist)

Original method to determine the list of non-kinked fibers. Tries to determine if a hit is either a non-kinked or kinked hit by using the distance of closest approach (*dca_min*) of the hits.

I.5 get_kink_list2(Clust,nMemb,nc,width,List,nlist)

Works in a similar manner as *get_kink_list_oldway*, but assumes (which is currently the case) that the pion fibers were ordered from decay vertex to the exit point. Not not consider a hit non-kinked if it is with the swath that has a width of *width*.

I.6 get_kink_list(Clust,nMemb,nc,width,List,nlist)

Called by *KinkFinder* . This routine assumes the pion fibers have already been order from decay vertex to the exit point. Therefore, the list of non-kinked fibers are 1 thru the number of non-kinked fibers.

I.7 pick_closest_track(...)

Inputs/Outputs: (Clust,nMemb,nc,d_trk1,d_trk2,min_dist,List1,nlist1,List2,nlist2,Both,nboth

This routine is was replaced by add_track_fibers. The purpose of this routine is to determine if a hit should be classified as a non-kinked fiber or a kinked fiber. This is accomplished by determining what track, *trk1 or trk2*, is closest to fiber *Clust(*,i,nc)*. The i^{th} hit is then stored into either $List1$ or $List2$. If the fiber is within the distance *min_dist*, then the i^{th} hit is stored in $List1$, $LIst2$, and $Both$ arrays.

I.8 add_track_fibers(...)

Inputs/Outputs: (Clust,nMemb,nc,d_trk1,d_trk2,min_dist,List1,nlist1,List2,nlist2,Both,nboth)

A more sophisticated version of *pick_closest_track*. The inputs and outputs are identical; only the 'guts' have changed.