HERA B Software Group

SUSi: a Pattern Recognition Algorithm for the HERA–B Silicon Vertex Detector

Jörg Rieling
MPI f. Kernphysik Heidelberg

January 1996

Contents

1 Intro

This software note describes *SUSi* (= **s**earching **u**tility for the **si**licon), a first pattern recognition program for the HERA–B Silicon vertex detector. *SUSi* uses a 3–dimensional track following method based on the Kalman filter. The development of this program was motivated by the following considerations:

1.1 Why All This ?

The strategy for the HERA–B farm reconstruction is to perform independent pattern recognitions both in front and after the magnet: *SUSi* is a first step towards the final Silicon reconstruction tool.* Given the need for a such a program, a second argument stresses the importance to start the development quite soon:
The years 94/95 have witnessed major changes in the layout of the Silicon detector. The old design combined 136 Silicon planes to 17 superlayers, each of which measured 2 sensitive coordinates ($0°, 90°$ or $45°, 135°$). In first studies, this geometry turned out to provide a good environment for pattern recognition. However, because of it's complicated mechanical construction a simpler layout had to be developed. Again 4 stereo angles were chosen ($0° \pm 2.5°, 90° \pm 2.5°$), but with 112 Silicon planes in only 7 superlayers the new design[1] places more detector modules in a certain Z–position. The question arises whether pattern recognition with a reasonable performance is possible in this case: it should be more difficult because the track information is essentially concentrated in three space points, whereas it was well distributed along the trajectory in the old design.

This note is a source code documentation of what has been developed so far; it intends to facilitate the efforts of those who want to start working with *SUSi* or other Silicon reconstruction tools. Another argument is certainly the comparative ease and safety in re–using already existing routines.

1.2 How to Get the Files

All necessary files are compiled on the `hydra.ifh.de` in the file:

`/net/sgi/herab/programs/jr/susi.tar.gz`

*However, this does not necessarily imply the final program to be just a refinement of *SUSi*. On the contrary, studying other approaches (e. g. track following in projections or global methods) might turn out to be very fruitful.

First, copy it into one of your directorys, then `gunzip` and unpack the archive file. You will now have a new subdirectory `./susi` where all necessary files are compiled:

1. Source code files `*.cmz`. *SUSi* and the merging package are provided as well as the appropriate *ARTE*[8], *PRISM*[6] and *TF* versions. The main programs `susi-main.f` and `merge-main.f`.
2. A `cmzlogon.kumac` which loads the necessary sequences and sets the compiler (at least for SGI's and HP's).
3. A makefile `mksgi` which compiles and links the files for the SGI's in Zeuthen.
4. Various data files (`hbu.dat, seedtab.dat, navtab.dat, ...`).
5. The subdirectories `./lib, ./libsgi, ./ftn, ./hb` and `./log` which are empty. They are used by CMZ and *ARTE*.
6. Last but nat least: a `BuildAll` shell script which performs the complete installation for the Zeuthen environment, and a `README` file that tells you in detail how to do that by your own.

Event and geometry files are not given because they are certainly too large. If you work in Zeuthen, use the files

`/net/sgi/herab/jenny/target.zeb`	(250 events)
`/net/sgi/herab/jenny/targetgeo.zeb`	(geometry file)

Otherwise you have to copy them onto your platform.

SUSi works at least with:

1. Version 1.03/07 of *ARTE*,*HBUTIL* and *HBCDES*
2. All *PRISM* versions
3. All `TF` versions

A new *ARTE* version is expected in these weeks. The migration to the new version should be no problem because all of *SUSi*s I/O is performed via *ARTE*–routines and functions which were only changed in their name.

2 Hit Generation

Using a set of given primary interactions and the distribution of matter in the detector, the MC simulation creates secondary particles and finally the raw data that would be produced in the real experimantal setup. This paragraph describes the MC's assumptions about primary reactions and detector geometry and concludes with an outline of the hit generation algorithm.

MC assumptions

The simulation uses $\bar{n} = 5$ Poisson distributed primary interactions to create hits in the detector. The number of primaries is thus higher than in the real detector which is projected to operate at $\bar{n} = 4$. One of those interactions is a golden B decay and 4 are mimum bias reactions. The primary vertices are distributed over the target structure: it consists of eight wires at four different Z – positions (see fig. 1 a.)). The momentum cutoff for secondaries is 1 MeV.
The geometry of the Silicon Vertex Detector has been modelled including the insensitive guard ring structure, the readout chips, the support arms and the enclosing Al – caps. The assumptions about radiation thickness are conservative; see fig. 1 b.) for details.

Hit Generation Algorithm

When a charged particle crosses a sensitive plane, GEANT produces a MIMP (= "Monte Carlo Impact Point"). The hit generation algorithm collects all MIMPs for a given detector plane and creates the raw data with certain assumptions about detector resolution σ, double hit resolution s_d, efficiency ε and noise level ε_{noise}. It proceeds as follows:

1. As a silicon strip plane measures only one dimension, the MIMPs are sorted in the measured coordinate m (cmp. 1.2.).

2. According to ε, a certain number of MIMPs is selected that will not give rise to a hit.

3. The MIMPs with sensitive coordinates m separated by less than s_d are used to create a single "double hit" which is in the middle between the two MIMPs.

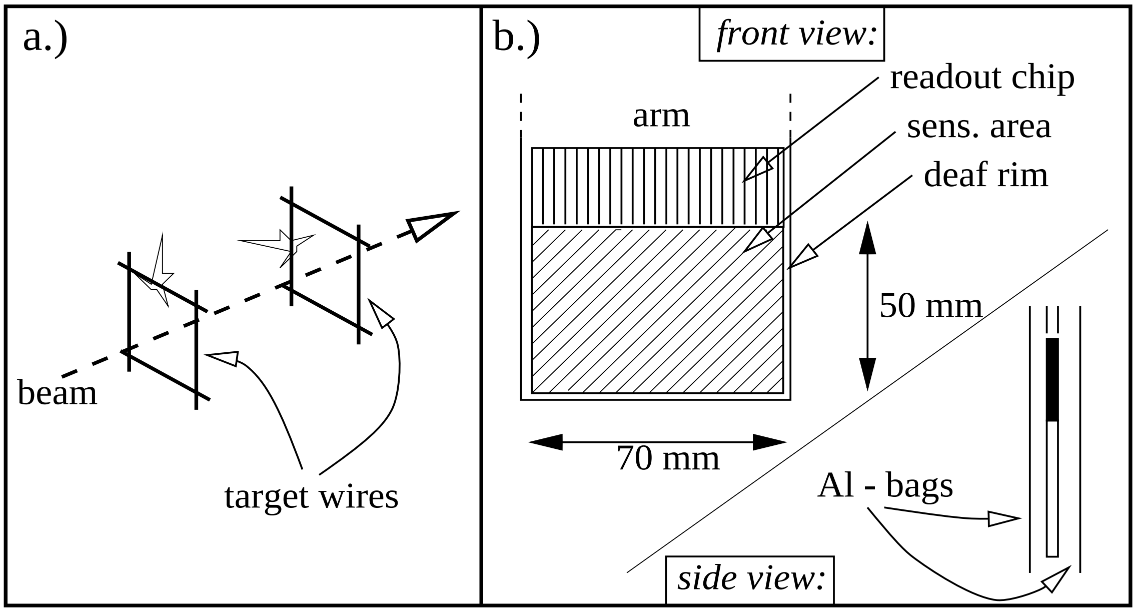

Figure 1: *Software model of the hardware configuration. a.) The target setup, b.) Front and side view of a silicon module. The front view does not show the enclosing Al–bags. The assumed distribution of matter is given in the table.*

component	*matter in XY–plane*
arm	*1.5* cm foam$(\frac{1}{16}\rho_C)$, *2.5 mm Be*
chips	$4 \times 300\,\mu m$ *Si, 1.5 mm Be*
Si–planes	$4 \times 250\,\mu m$ *Si, 1.5 mm Be*
Al–bag	$2 \times 125\,\mu m$ *Al*

4. The sensitive coordinates of the hits are smeared according to a Gaussian distribution with width σ.[†]

5. Finally a number of uniformly distributed noise hits is added.

The raw data for a certain detector plane is thus a set of measured coordinates — the creation of "digits" which correspond to the different pulse heights sampled for each silicon strip is foreseen but not yet implemented.
Fig. 2 shows a typical raw data event display; it is the task of the pattern recognition algorithm to reconstruct the particle tracks that gave rise to this line pattern.

[†]For the hits from the merged MIMPs the position resolution is thus $\sigma_{melt} = s_d/\sqrt{12} \oplus \sigma$, if a uniform distribution in $\Delta m = m_1 - m_2$ is provided for $|\Delta m| \leq s_d$.

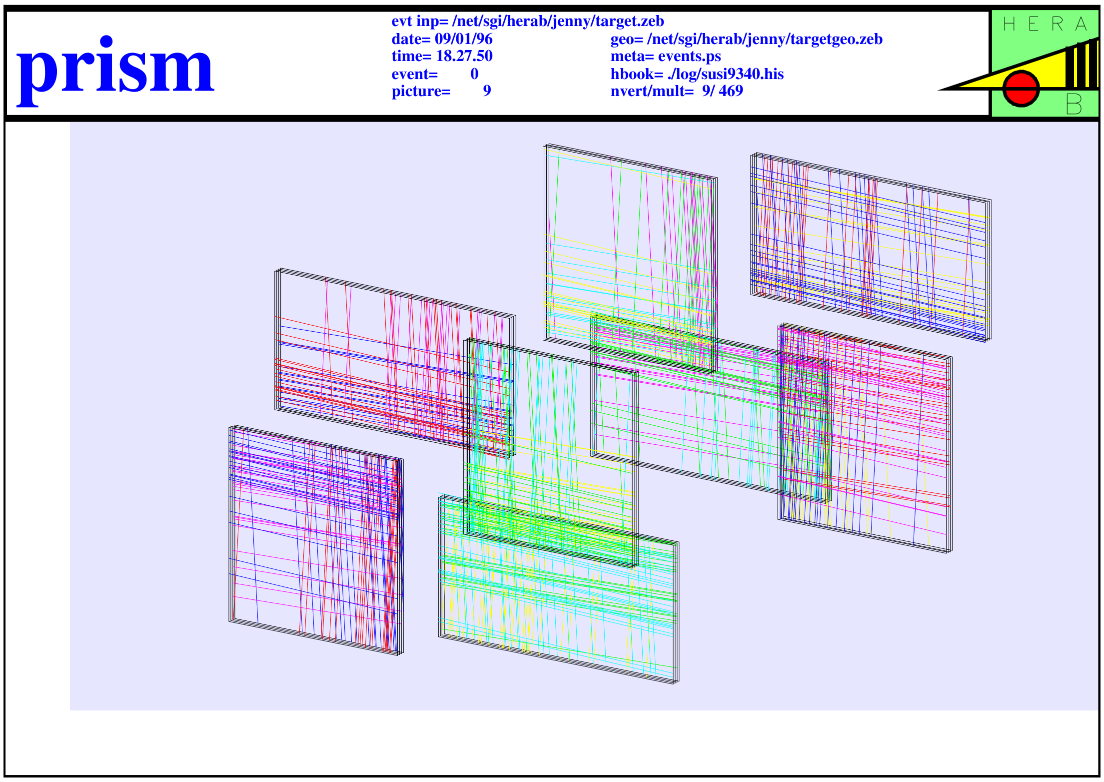

Figure 2: *Raw data event display from the superlayers 3 and 4. The fine lines inside the silicon detector planes symbolize a couple of fired strips (a "cluster").*

3 Track Finding

A particle traversing an ideal detector creates a hit in every sensitive plane it touches — in absence of a magnetic field these hits are arranged on a straight line. However, in reality the detector's finite detecting probability and limited position resolution spoil the trace of the projectile; and as it experiences multiple scattering, it's path is no longer a straight line. Moreover, the presence of many particles makes it often impossible to decide to which a certain hit belongs to.
Given the set of hits in a detector, it is now the task of the track finding algorithm to collect all hits that have been created by a certain particle. Track finding methods can be subdivided into *global* and *local* methods. Global methods treat all measurements in the same way: therefore they are unbiased. On the contrary, local methods deal with only one track candidate at a time, typically by starting with a small set of "seed" measurements. Because of this initialization, local methods are never unbiased.
The calculation time tends to be smaller for global methods; on the other hand, local algorithms proved to be quite robust even at high particle densities. Furthermore, they allow to include multiple scattering in a very natural way. For these reasons a 3–dimensional track follwing algorithm based on the Kalman filter has been chosen as the first step towards the Silicon track reconstruction for HERA–B. The next paragraph summarizes the necessary geometry conventions, and the two following sections desribe the implemented pattern recognition algorithm in detail.

3.1 Conventions

Coordinate System. The origin of the coordinte system is in the center of the beam at the position of the downstream target frame (see fig. 3 a.)). $\vec{e}_z$ is parallel to the beam, $\vec{e}_x$ points towards the center of the HERA–ring and $\vec{e}_y$ points upwards.

Measurement Plane. As the sensitive planes of the Silicon detector are neither rotated around $\vec{e}_z$ nor around a vector in the XY–plane, their position and orientation is completely determined by the Z–position z_{pl} and the 4 numbers $x_{min}, x_{max}, y_{min}, y_{max}$ (see fig. 3 b.)). The stereo angle α_{st} is the angle between the silicon strips and $\vec{e}_y$; it is counted mathematically negative.
Given a plane with α_{st}, the *local coordinate* of a vector (x,y) is defined as

$$m \stackrel{def}{=} x \cos \alpha_{st} - y \sin \alpha_{st}.$$

m is the (signed) projection of (x,y) on the direction perpendicular to the strips.

Track parameters. As there is no field in the Silicon Detector, a track can be parametrized by the parameter vector

$$p(z_0) = \begin{pmatrix} x(z_0) \\ y(z_0) \\ m_x \\ m_y \end{pmatrix},$$

where x and y depend on the position of evaluation. m_x, m_y are the slopes in X and Y. We denote the covariance matrix with $\mathcal{C} = \mathrm{cov}(\vec{p}(z_0))$.

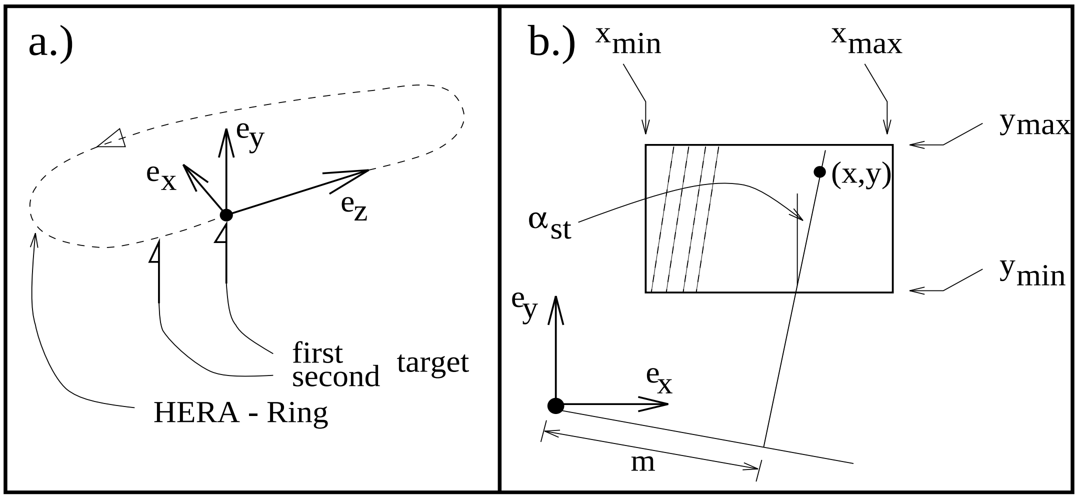

Figure 3: *a.) Orientation and origin of the coordinate system. b.) sketch of geometrical quantities at a measurement plane.*

3.2 Pattern Recognition Algorithms

The reconstruction of tracks with a track follwing method requires 3 major steps: at first the **seeding step** generates initial parameter vectors $\vec{p}_0$ by combining detector hits in a systematic way. The number of these *seeding vectors* must be very high in order to catch all tracks of the physical particles.
The **propagation step** moves the vectors through the detector where they collect

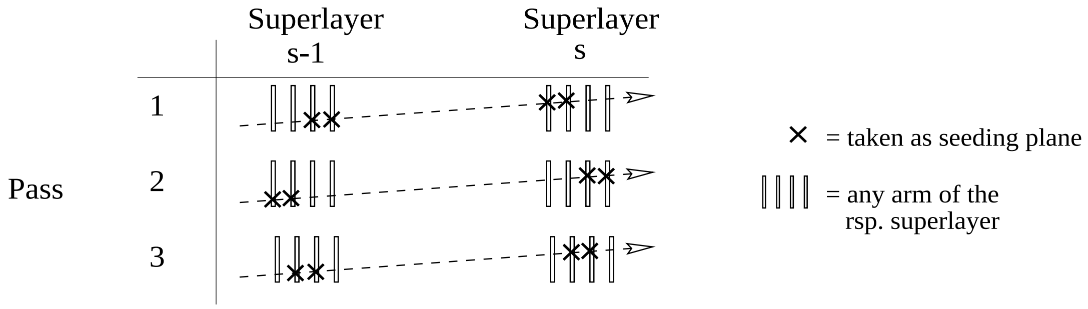

Figure 4: *Seeding scheme used in the PR algorithm. The numbers 1,2,3 denote the different passes.*

more hits if they find any along their path — vectors with too few hits are rejected, so that this step leaves only a small number of seeds. The group of hits belonging to one of the remaining seeding vectors is called *track candidate*; they are often incompatible with each other because they have too many hits in common.
For this reason the **selection step** finds all incompatible candidates and selects only the best ones: they form the set of *reconstructed tracks.*

3.2.1 The Seeding Step

As track vectors are 4–dimensional, we need 4 measurements to determine an initial vector. By selecting groups of 4 sensitive planes and combining their hits, one obtains the seeding vectors $\{\vec{p}_0\}$. In terms of a computer language this corresponds to five nested loops: one over the plane groups and four over the hits in each plane. When defining the plane groups (="seeding plane combinations"), a few points have to be considered:

- If a particle which is interesting from the physical point of view is inside the geometrical acceptance, it must hit a seeding plane combination.

- Given a set of 4 planes, it may well be that an intersecting particle does not produce a hit in one of them ($P_{4hits} = \varepsilon_{pl}^4!$). It is therefore not sufficient to cover a certain region in the detector with only a single seeding combination.

- To minimize the variance of $\vec{p}_0$, the seeding planes should be sufficiently separated.

A schematic drawing of the seeding scheme that has been implemented can be found in fig. 4. It is possible to treat the whole detector at once in each pass

by varying the plane combinations systematically: first hold s constant and treat every arm combination that could have been intersected by a particle. Then vary s and repeat the first step.
If now a track was not caught in the first *pass* because of missing hits, it is most probable that it will be found in the second or third. Assuming $\varepsilon_{pl} = 0.95$ an analytical calculation shows that the probability to catch a track with this scheme is $P_{found} = 97.1\%$.
Because the arms of the Silicon detector overlap in a non–trivial way, a dedicated algorithm has been written that generates the seeding plane combinations for all superlayer indices s, passes and particle directions: it scans the detector with "rays" originating on the target structure and memorizes the combinations of intersected planes that match the above scheme. For each pass the combinations are then sorted in the Z–position of their last plane in decreasing order and are finally written to a file (*seeding table*). The seeding table is read by the setup routine of the pattern recognition program.

Now, when the pattern recognition algorithm knows about the seeding combinations, it has to calculate the track parameters from the plane geometries and the measurement values.
If we mark the planes with $i = 1 \ldots 4$ we have

$$\begin{aligned}
m_i &= x(z_i)\cos\alpha_{st,i} - y(z_i)\sin\alpha_{st,i} \\
&= p_1\cos\alpha_{st,i} - p_2\sin\alpha_{st,i} + p_3 z_i\cos\alpha_{st,i} - p_4 z_i\sin\alpha_{st,i} \\
&= (\cos\alpha_{st,i}, -\sin\alpha_{st,i}, z_i\cos\alpha_{st,i}, -z_i\sin\alpha_{st,i})^T\cdot\vec{p}, \\
\vec{m} &= \mathcal{M}\,\vec{p},
\end{aligned}$$

where $\vec{p}$ is evaluated at $z = 0$. Analytical inversion yields

$$\vec{p} = \mathcal{M}^{-1}\vec{m},$$

with

$$\begin{aligned}
\mathcal{M}^{-1} &= f^{-1}\,\mathcal{P}, \quad \text{where:} \\
\mathcal{P}_{11} &= c_3 s_2 s_4 z_3 \Delta z_{24} + c_2 s_3 s_4 z_2 \Delta z_{43} + c_4 s_2 s_3 z_4 \Delta z_{32} \\
\mathcal{P}_{12} &= c_1 s_3 s_4 z_1 \Delta z_{34} + c_4 s_1 s_3 z_4 \Delta z_{13} + c_3 s_1 s_4 z_3 \Delta z_{41} \\
\mathcal{P}_{13} &= c_2 s_1 s_4 z_2 \Delta z_{14} + c_1 s_2 s_4 z_1 \Delta z_{42} + c_4 s_1 s_2 z_4 \Delta z_{21} \\
\mathcal{P}_{14} &= c_1 s_2 s_3 z_1 \Delta z_{23} + c_3 s_1 s_2 z_3 \Delta z_{12} + c_2 s_1 s_3 z_2 \Delta z_{31}
\end{aligned}$$

$$
\begin{aligned}
-\mathcal{P}_{21} &= c_2c_4s_3z_3\Delta z_{24} + c_3c_4s_2z_2\Delta z_{43} + c_2c_3s_4z_4\Delta z_{32} \\
-\mathcal{P}_{22} &= c_3c_4s_1z_1\Delta z_{34} + c_1c_3s_4z_4\Delta z_{13} + c_1c_4s_3z_3\Delta z_{41} \\
-\mathcal{P}_{23} &= c_1c_4s_2z_2\Delta z_{14} + c_2c_4s_1z_1\Delta z_{42} + c_1c_2s_4z_4\Delta z_{21} \\
-\mathcal{P}_{24} &= c_2c_3s_1z_1\Delta z_{23} + c_1c_2s_3z_3\Delta z_{12} + c_1c_3s_2z_2\Delta z_{31} \\
\mathcal{P}_{31} &= c_4s_2s_3\Delta z_{23} + c_2s_3s_4\Delta z_{34} + c_3s_2s_4\Delta z_{42} \\
\mathcal{P}_{32} &= c_3s_1s_4\Delta z_{14} + c_4s_1s_3\Delta z_{31} + c_1s_3s_4\Delta z_{43} \\
\mathcal{P}_{33} &= c_4s_1s_2\Delta z_{12} + c_1s_2s_4\Delta z_{24} + c_2s_1s_4\Delta z_{41} \\
\mathcal{P}_{34} &= c_2s_1s_3\Delta z_{13} + c_3s_1s_2\Delta z_{21} + c_1s_2s_3\Delta z_{32} \\
-\mathcal{P}_{41} &= c_2c_4s_3\Delta z_{42} + c_2c_3s_4\Delta z_{23} + c_3c_4s_2\Delta z_{34} \\
-\mathcal{P}_{42} &= c_1c_4s_3\Delta z_{14} + c_1c_3s_4\Delta z_{31} + c_3c_4s_1\Delta z_{43} \\
-\mathcal{P}_{43} &= c_1c_2s_4\Delta z_{12} + c_2c_4s_1\Delta z_{24} + c_1c_4s_2\Delta z_{41} \\
-\mathcal{P}_{44} &= c_1c_3s_2\Delta z_{13} + c_1c_2s_3\Delta z_{21} + c_2c_3s_1\Delta z_{32} \\
f &= -(c_1c_2\, s_3s_4\, \Delta z_{12}\Delta z_{43} + c_1c_3\, s_2s_4\, \Delta z_{13}\Delta z_{24} + \\
&\qquad c_1c_4\, s_2s_3\, \Delta z_{14}\Delta z_{32} + c_2c_3\, s_1s_4\, \Delta z_{23}\Delta z_{41} + \\
&\qquad c_2c_4\, s_1s_3\, \Delta z_{24}\Delta z_{13} + c_3c_4\, s_1s_2\, \Delta z_{34}\Delta z_{21}) \\
c_i &\stackrel{def}{=} \cos\alpha_{st,i} \\
s_i &\stackrel{def}{=} \sin\alpha_{st,i} \\
\Delta z_{ij} &\stackrel{def}{=} z_i - z_j\,.
\end{aligned}
$$

The matrix $\mathcal{M}^{-1}$ does only depend on the geometry data of the seeding planes. It is therefore calculated once and forever for each seeding combination in the program setup: the determination of the starting vector is hence a simple matrix–vector–multiplication.

3.2.2 The Propagation Step

Let us assume that the algorithm is now able to construct seeds by the method described in the previous section. The task is now to loop over the seeds and to check if there are some more hits on the line defined by the seeding vectors. This is done by propagating and updating the vectors until no further hits are found (meaning the particle has been created within the detector planes) or the track leaves the detector (particle was created outside detector).
As stated above, it is necessary to treat the whole detector more than once. For

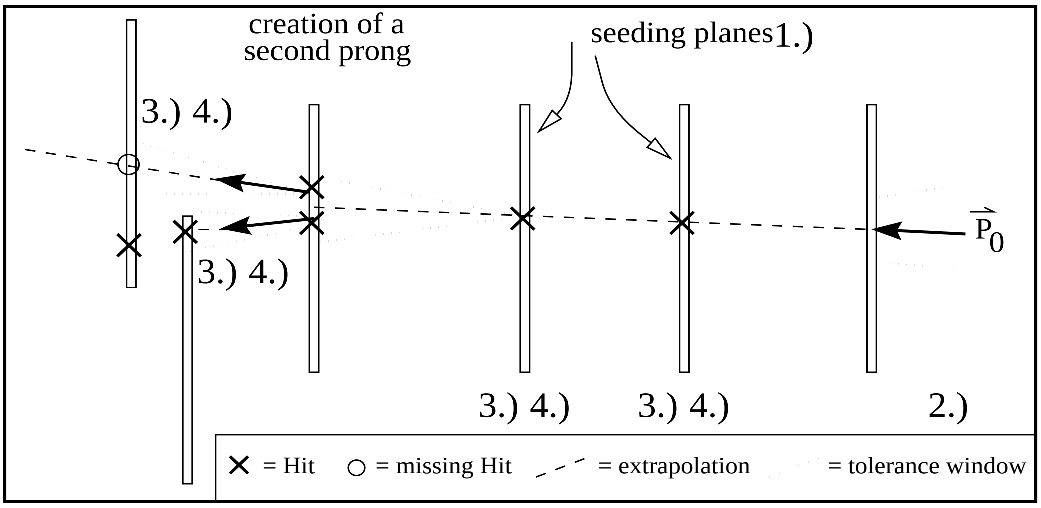

Figure 5: *Track initialization and propagation in 2 dimensions. The numbers correspond to the rsp. steps described in the text.*

each pass the algorithm proceeds as follows (compare fig. 5):

(...)

0.) Get the 4 seeding measurements.

- **1.)** Calculate seeding vector $\vec{p_0}$ from the 4 measurements. Set $\mathcal{C} = \infty$. Then, start the track following at high Z-positions and proceed towards the target:
- **2.)** Extrapolate $\vec{p_0}$ to the last plane in the downstream seeding arm. If you don't find a hit, ignore (=proceed anyway). If you find at least one hit, create a track prong (="subtrack" that stems from $\vec{p}_0$) for each hit and udate $\vec{p_0}$ and $\mathcal{C}$ with the rsp. measurement. — Now, perform the following 2 steps for every prong:
  - **3.)** find the next intersected sensitive plane and propagate $\vec{p_0}$, $\mathcal{C}$ to it. If there is no next plane (= particle left detector) then treat next prong.
  - **4.)** If this plane is a seeding plane, then update $\vec{p_0}$, $\mathcal{C}$ with the seeding hit. Otherwise try to find hits in the tolerance region. If you don't find a hit, decide if there were too many ($> n_{max,miss}$) faults in

sequence: if so, treat the next prong, otherwise ignore. If you find one hit, update $\vec{p_0}$, $\mathcal{C}$ with this measurement. If you find more than one hit, create additional prongs and update the $\vec{p_0}$'s, $\mathcal{C}$'s with the rsp. measurements. Process these prongs with **3.)**, **4.)**.

5.) You have now a set of complete prongs. Discard all prongs that do not pass certain cuts. Select the best remaining prong (if existing): store it as the "track candidate" resulting from the current seed, then

return to 0.) if there are any seeds left.

(...)

It is worthwhile to comment upon the various steps a little more in detail:

Kalman formalism. The track parameter propagation and updating in **2.), 3.)** and **4.)** has been installed using the Kalman formalism. This note confines itself to the actually implemented formulas; more detailed discussions ([2][3][4]) lead to a variety of applications using this flexible method.
The propagation equations transport the track parameter from z_i to z_f:

$$\vec{p}_i,\ \mathcal{C}_i \ \longmapsto\ \vec{p}_f = \mathcal{F}_{if}\,\vec{p}_i,\ \mathcal{C}_f = \mathcal{F}_{if}\,\mathcal{C}_i\,\mathcal{F}_{if}^T,\ \text{where}$$

$$\mathcal{F}_{if} = \begin{pmatrix} 1 & 0 & z_i - z_f & 0 \\ 0 & 1 & 0 & z_i - z_f \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

A very easy and fast method to include the multiple scattering contribution of the plane at z_i is to simply add the mean MS angle $\vartheta_{ms} = 0.0136\,[\mathrm{GeV}]/\beta cp \times \sqrt{x/X_0}$ to $\mathcal{C}_{f,33}$ and $\mathcal{C}_{f,44}$. However, as MS introduces correlations between the angles([5],citerainer2), this method is only a (good) approximation. Nevertheless it is used in the algorithm because it is advantageous for the speed of the program.—
In the Silicon there is no information about the momentum of the track. For this reason the program assumes that $p = {\scriptstyle const} = \bar{p}$ for all tracks, where $\bar{p}^{-1} \stackrel{def}{=} < \frac{1}{p} >$.
The mean is taken over all reconstructed tracks.
In **2.)** and **4.)** a hit is added to a track prong: $\vec{p}$ and $\mathcal{C}$ which carry the track information of all hits so far are updated or "filtered" with the new measurement

m. The filter equations are

$$\begin{aligned}\vec{p}_{f,filt} &= \vec{p}_f + \mathcal{K}\,(m - \mathcal{H}\,\vec{p}_f) \\ \mathcal{C}_{f,filt} &= (1 - \mathcal{K}\,\mathcal{H})\,\mathcal{C}_f, \text{ where}\end{aligned}$$

$$\begin{aligned}\mathcal{K} &\stackrel{def}{=} (\sigma_f^2 + \mathcal{H}\mathcal{C}_f\,\mathcal{H}^T)^{-1} \cdot \mathcal{C}_f\,\mathcal{H}^T && (\text{"Kalman gain matrix"}) \\ \mathcal{H} &\stackrel{def}{=} (\cos\alpha_{st,f},\ -\sin\alpha_{st,f},\ 0,\ 0) && (\text{projector on sens. coordinate})\end{aligned}$$

The χ^2–contribution of the filtering step is

$$\begin{aligned}\Delta\chi^2 &= r^2/R \\ r &\stackrel{def}{=} m - \mathcal{H}\,\vec{p}_f && (\text{residual}) \\ R &\stackrel{def}{=} (1 - \mathcal{H}\mathcal{K})\sigma_f^2 && (\text{measurement cov. matrix}).\end{aligned}$$

Tolerance regions for hit searching. When a vector $\vec{p}$ has been propagated to an adjacent plane, the steps **2.)** and **4.)** determine if there are any hits in a tolerance region $[m_{pred} - \Delta m,\ m_{pred} + \Delta m]$ around an estimated local coordinate m_{pred}. It turned out to be most favourable to introduce a simple tolerance function that uses the Z–distance to the last measurement of the current local coordinate:

$$\Delta m = \begin{cases} \Delta m_{min} & \text{for } (z_{last} - z)t_m \leq \Delta m_{min} \\ (z_{last} - z)t_m & \text{for } \Delta m_{min} < (z_{last} - z)t_m < \Delta m_{max} \\ \Delta m_{max} & \text{for } (z_{last} - z)t_m \geq \Delta m_{max} \end{cases}$$

The algorithm constants m_{min}, t_m, m_{max} have to be determined empirically.

Navigation Tables. As a particle traversing a given plane x can only intersect a small number of adjacent planes $n(x)$, it would be a waste of time if the algorithm checked every upstream plane if it was touched. The navigation table contains the set of possible *next planes* for all x; the sets are sorted in decreasing Z. The table is generated by a dedicated algorithm similar to the one that produces the seeding table. It is read by the pattern recognition setup routine.

Track Quality and Cuts. When the steps **1.)** to **4.)** have finished to collect hits for the prongs, step **5.)** applies absolute and relative cuts so that at most one prong is left as a track candidate.
Absolute cuts: Firstly a prong must have a certain minimum number of hits.

Secondly, the number of missing hits may not exceed a value $n_{max,\, miss}(n_{th})$ that depends on the theoretical number of hits $n_{th} = n_{Hit} + n_{miss}$. The function $n_{max,\, miss}$ has been determined by demanding that the probability to have at least n_{miss} faults in n_{th} planes is smaller than 1%. Using the binomial distribution, $n_{max,\, miss}(n_{th})$ can easily be calculated. Finally, the track quality

$$Q = n_{Hit} - f_{miss} n_{miss} - f_{\chi^2} \chi^2$$

may not be below a minimum value Q_{min}. Again, the values for f_{miss} and f_{χ^2} were fixed empirically. Any prong that does not meet the above requirements is discarded.

Prong comparison: The remaining prongs are compared in their quality Q, and only the best one (=highest Q) is accepted as the track candidate resulting from the above seeding combination.

This is how track candidates are created for a single pass. But before the detector is examined a second and third time, candidate selection and hit marking take place.

3.2.3 The Selection Step

The pattern recognition program has now found a number of track candidates that should include the real tracks. Those candidates are often "incompatible" with each other because they have too many common hits. Two tracks are called incompatible if

$$p_{comm} = \frac{2\, n_{comm}}{n_{Hit,1} + n_{Hit,2}} \geq p_{max} = \begin{array}{l}\text{max. allowed degree}\\ \text{of equivalence}\end{array} = 0.3,$$

where $n_{Hit,i}$ denotes the number of hits for candidate i. Many common hits can for example result from two seeds that evolve into the same track or from two particles travelling closely together. —

The incompatible track pairs now have to be located, and the tracks which most likely correspond to a physical particle should be selected: they are the *reconstructed tracks* from the current pass. Two different routines have been written for this purpose:

1. One–by–one comparison. Here the algorithm calculates successively p_{comm} for all pairs containing a certain reconstructed track. If one pair turns out to

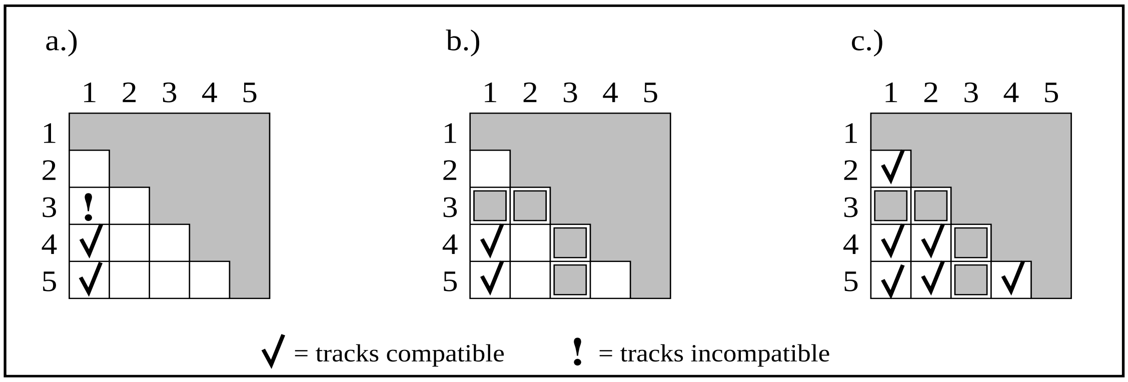

Figure 6: *Example for the one–by–one comparison for 5 track candidates. The cand.'s 1 and 3 are incompatible. Track 3 has the lower* Q.

be incompatible, the track candidate with lower Q is removed (see fig. 6). The algorithm terminates if all remaining pairs have been examined. Note that a different succession of candidates may result in different reconstructed tracks: the method is biased.

2. All–to–all comparison. First p_{comm} is calculated for *all* pairs. The pairs are then sorted in their degree of equivalence. Starting at the lowest p_{comm}, one track per incompatible pair is removed. This algorithm is unbiased.

The second method is in principle superior to the first one because it is unbiased and uses the full information. However, in practice they have the same selection power — for the sake of speed the one–by–one comparison is used.

3.2.4 Hit Marking

It is necessary to considerably reduce the calculation time in the following passes; a lot of CPU time is spent to treat the enormous number of seeding combinations ($n_{seeds} \sim n^4_{tracks}$!). Hence, removing the hits of the reconstructed tracks is very effective. On the other hand, applying a removal scheme always implies the loss of information.
In such cases of quandary it proved to be a good compromise to mark the used hits and not to combine them to further seeds. But if they are found during the propagation of a vector backwards they will be accepted.

3.3 Analysis

This section gives a survey of the existing analysis and diagnosis algorithms. As the methods are not always matter–of–course, the underlying concepts and formulas are compiled.

3.3.1 Track Recognition Probabilities

Of course it is of central importance to check the reconstruction performance of the programs not only by eyescan but also using formally well–defined quantities. Especially the recognition power for particles that should be found theoretically is of high interest:

Proper Tracks. A Monte–Carlo particle is called *proper* if it fulfills the following demands:

1. It has at least one hit in at least three Silicon superlayers.
2. The momentum is $\geq 2\,\mathrm{GeV}$.
3. The particle is not a γ–daughter, i. e. due to photon conversion.

The $1^{\underline{\mathrm{st}}}$ requirement defines the geometrical efficiency, the other ones exclude highly scattered particles and those which are irrelevant for physics analysis.

Reconstruction criterion. Eventually the pattern recognition algorithms will associate hits that have been created by different particles. A reconstructed track is now called *correct* if the fraction of the hits that are due to a single particle exceeds a threshold:

$$\text{track correct} \overset{def}{\Longleftrightarrow} f_{uniq} = \frac{n_{uniq}}{n} \geq f_{min} = 0.7\,,$$

where n denotes the number of hits of the (reconstructed) track. Of course, a double hit counts for more than one MC particle.
The number of hits belonging to the MC particle n_{mc} defines the *hit finding probability* $\varepsilon_{Hit} \overset{def}{=} n_{uniq}/n_{mc}$.

Recognition probabilities. The *efficiencies* relate the reconstructed tracks to the "real" tracks that constitute the event: the probability to (correctly) reconstruct a proper track in a given event i is called *single event efficiency* $\varepsilon_i \stackrel{def}{=} n^{rec}_{pr,i} / n_{pr,i}$. $n^{rec}_{pr,i}$ gives the number of reconst. proper tracks. A measure for the quality of the program is the *mean efficiency*

$$\bar{\varepsilon} \stackrel{def}{=} \frac{\sum \varepsilon_i}{N}.$$

N denotes the number of processed events. Another quality criterion is the rate of fake tracks or "ghosts". Similiar to the above, the *ghost efficiency* is defined as

$$\bar{\varepsilon}_{gh} \stackrel{def}{=} \frac{\sum \frac{n_{false,i}}{n_i}}{N}.$$

$n_{false,i}$ are the incorrectly reconstructed tracks, and n_i gives the total number of reconst. tracks.
The *extra efficiency* $\bar{\varepsilon}_x$ is the rate of correctly recognized non–proper tracks:

$$\bar{\varepsilon}_x \stackrel{def}{=} \frac{\sum \frac{n_{x,i}}{n_i}}{N}$$

and is thus a measure of how restrictive the proper track selection is.
The above efficiencies are calculated both for all tracks as well as for special subsets (e. g. the leptons and pions from the golden decay).

3.3.2 Track resolution

In order to find out how precise the origin of a particle can be determined, the analysis calculates the *vertex residual* $\Delta\vec{x}$, which is the distance of the extrapolated reconstructed track to it's vertex:

$$\Delta\vec{x} = \begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix} \stackrel{def}{=} \tilde{\vec{x}}_{Vtx} - [\,(z - \tilde{z}_{Vtx}) \begin{pmatrix} p_3(z) \\ p_4(z) \end{pmatrix} + \begin{pmatrix} p_1(z) \\ p_2(z) \end{pmatrix}].$$

$\vec{p}$ is the parameter vector of the reconstructed particle at an arbitrary position z, and $\tilde{\vec{x}}_{Vtx}$ and $\tilde{z}$ denote the true coordinates of the production vertex.
The vertex resolution $\mathrm{cov}(\vec{x}_{Vtx})$ is of course related to the track parameter resolution $\mathrm{cov}(\vec{p})$ in a way that depends on the topology. Hence an exact determination requires a vertex fit ([2][5]) for each event. However, the current program version does not offer vertexing routines because the $\Delta\vec{x}$–distributions contain sufficient information to estimate the resolution.

3.4 Algorithm Diagnosis

A few instruments have been developed to understand the pattern recognition program's innermost life in more detail. They include the monitoring of all kinds of basic quantities (e. g. how often does a given seeding combination produce a track? how efficient is the selection step?) but also some more sophisticated tools (What is the probability to reconstruct a track that is not caught by the seeding scheme, or has two faults in sequence, or both?). For comprehensive information please consult the index.

4 Track Merging

The actual basic idea of the HERA–B track reconstruction is to perform independent pattern recognitions in both the Silicon and the main tracking area behind the magnet. This means a certain level of redundancy is produced which makes it more certain that the raw data can be disentangled even at high multiplicities or if isolated detector modules fail. Consequently, methods must be developed which enable us to really take advantage from the extra information.
A first step towards this goal is to relate the reconstructed tracks from the Silicon to those from the main tracking. In short, the *track merging* works as follows:

1. Propagate all reconst. tracks from the silicon and the main tracking to a reference surface (= the XY–plane at $z_{exit} = z$ of downstream surface of the silicon vertex tank's exit window).

2. Merge track segments to segment pairs. Discard pairs with too high merging χ^2. Accept pairs with least χ^2's.

Up to now algorithms which perform the track melting with good results have been written. What is left to be developed are methods which treat the remaining track segments: by this means it should be possible to further enhance the recognition efficiency ε for the whole detector.
This section describes at first the propagation step 1.) and then outlines two different merging strategies (2.)).

4.1 Silicon Refit

To obtain $\vec{p}$–vectors of optimal precision on the exit window, three different methods were tested. As HBGEAN unfortunately does not deliver the intersection vectors of the MC particles with a reference plane (...so far! :-)), the quality criterion for the different methods was the variance of the residual vector $\Delta\vec{p}(z) = \vec{p}_{refit}(z) - \vec{p}_{mimp}(z)$, where z is the position of the track's most downstream hit.

Simple back–extrapolation. After the last measurement the track parameters $\vec{p}(z_{last})$, $\mathcal{C}(z_{last})$ contain the information from all hits — moreover, $\vec{p}$ is of minimum variance among all linear estimators[2]. A very simple method with acceptable

results is thus to prolong $\vec{p}(z_{last})$ onto the exit window and to use an effective cov. matrix $\mathcal{C}(z_{last}) = {\scriptstyle const.}$ However, this "quick & dirty" technique does not consider the slight curvature of the trajectory which is due to multiple scattering.
Unlike the previous one, the following two algorithms pay attention to the actual distribution of dead matter (arms, chips, Al–bags).

Kalman Smoothing. The probably most general tool to perform a refit is the *Kalman smoother*[2]. It allows to determine the optimal track parameters at any point along the path by the following recursion relations:

$$\begin{array}{lcll} \vec{p}_k^n & = & \vec{p}_k^k + \mathcal{A}_k(\vec{p}_{k+1}^n - \vec{p}_{k+1}^k) & \text{smoothed parameter vector} \\ \mathcal{C}_k^n & = & \mathcal{C}_k^k + \mathcal{A}_k\,(\mathcal{C}_{k+1}^n - \mathcal{C}_{k+1}^k)\mathcal{A}_k^T & \text{sm'd cov. matrix} \\ \delta\chi_{n,k}^2 & = & (\vec{r}_k^n)^T (R_k^n)^{-1}\,\vec{r}_k^n & \text{sm'd } \chi^2\text{–contribution,} \end{array}$$

where

$$\begin{array}{lcll} \vec{r}_k^n & = & m_k - \mathcal{H}_k \vec{p}_k^n & \text{sm'd residual} \\ R_k^n & = & \sigma_k - \mathcal{H}_k\,\mathcal{C}_k^n\,\mathcal{H}_k^T & \text{sm'd measurement cov. matrix} \\ \mathcal{A}_k & \stackrel{def}{=} & \mathcal{C}_k^k \mathcal{F}_k^T (\mathcal{C}_{k+1}^k)^{-1} & \text{"smoother gain matrix"} \end{array}$$

x_k^n, $(x = \vec{p}, \mathcal{C}, \delta\chi^2, r, R)$ denotes the rsp. quantity at the position z_k if one uses the information from all measurements up to z_n. $\{x_k^k \mid k \leq n\}$ is thus the *history* of the filtered quantities from the Kalman steps, and $\{x_{k+1}^k \mid k < n\}$ is the history of predictions.— Obviously the algorithm starts with $k = n$ and delivers $x_{n-1}^n, x_{n-2}^n, \cdots, x_1^n$ successively. Note that the $\delta\chi^2$'s are meaningful only now: an effective outlier rejection during the smoothing is (in principle) possible.
In practice the Kalman smoother turns out to be a little bit problematic because it is very sensitive to incorrect cov. matrices: especially the inversion amplifies slight inconsistencies. Now, as $\mathcal{C}$ is *a priori* poorly known in the vertex detector because there is no momentum information, this results in a still sub–optimal $\Delta\vec{p}(z_{last})$–resolution.

Kalman refit. This method is again straightforward. Simply set $\vec{p}_0 = \vec{p}(z_{last})$ and $\mathcal{C}_0 = \mathcal{C}(z_{last})$ and propagate downstream. Add MS contributions every time you cross matter.

The Kalman refit delivers the optimal track parameters only at the last measurement it processes: if one is only interested in the the values that $\vec{p}, \mathcal{C}$ have

on the reference surface, this approach is analytically equivalent to the Kalman smoother. But measurements along the trajectory have distorted χ^2–contributions — it is therefore not possible to implement a reliable outlier removal.
On the other hand, the Kalman refit is numerically more stable and requires less calculation steps, especially no matrix inversion. This results in a better $\Delta\vec{p}$–resolution. For this reason the last method is used as the Silicon refit.[‡]

4.2 Main Tracking Refit

When the track finding in the main pattern recgnition area is finished, the track parameters of the found tracks are extrapolated to the magnet. The subsequent propagation routines perform an "ideal pattern recognition" in the current implementation: using only the hits that stem from the rsp. MC particle, they transport $\vec{p}$, $\mathcal{C}$ on helix segments through the field. The actually implemented formula and a short discussion of some typical problems can be found in [7]. When the track segments have finally reached the reference plane, they are merged with those coming from the Silicon.

4.3 The Merging Step

If one wants to merge two vectors $\vec{p}_1$ and $\vec{p}_2$ with errors $\mathcal{C}_1$, $\mathcal{C}_2$ into a single vector $<\vec{p}>$, the χ^2–*ansatz* is written:

$$\chi^2 = \Delta\vec{p}_1^{\,T}\,\mathcal{C}_1\,\Delta\vec{p}_1 + \Delta\vec{p}_2^{\,T}\,\mathcal{C}_2\,\Delta\vec{p}_2\,,$$

where $\Delta\vec{p}_i \stackrel{def}{=} \vec{p}_i - <\vec{p}>$. Minimization leads to

$$<\vec{p}>_{min} = (\mathcal{C}_1^{-1} + \mathcal{C}_2^{-1})^{-1}(\mathcal{C}_1^{-1}\,\vec{p}_1 + \mathcal{C}_2^{-1}\,\vec{p}_2)\,,$$

which is simply the weighted mean. The χ^2 is used to test the hypothesis of two track segments belonging to each other. To determine the best association relation between the Main tracks and the Silicon tracks, two algorithms were written which have some similarity with the candidate selection routines (compare page 14).

[‡] An outlier removal scheme based on the smoother is subject to further studies. A subsequent Kalman refit would deliver even better resolutions.

Next Neighbour Method. Apart from slight differences, this technique corresponds to the one–by–one comparison:

- Select a non–marked Silicon vector and perform the following 2 steps:

    1. Loop over all non–marked Main vectors and calculate the χ^2 for each pair. Discard pairs with $\chi^2 \geq \chi^2_{max} \stackrel{def}{=} 500$.
    2. Accept the best remaining pair (="next neighbour", i. e. least χ^2) and mark both the Silicon and the Main segment.

- Repeat until you treated all Si–segments.

The resulting association relation depends in principle on the order in which the track vectors enter: the algorithm is biased.

Association by Sorting. Clearly, this algorithm corresponds to the all–to–all candidate selection: First, the χ^2 is calculated for all pairs. The pairs are then sorted in χ^2, and those with $\chi^2 > \chi^2_{max}$ are removed. Starting from the lowest χ^2, the pairs are accepted if none of the rsp. segments is marked — in case a pair is accepted, it's segments are marked. This method is unbiased.

Both algorithms prove to work quite well and have the same association power. Due to calculation time the simpler next neighbour method is used.

4.4 Analysis

The book–keeping is a little bit more complicated for the merging algorithm because two sorts of proper tracks are involved. Furthermore, it is not certain that all found tracks can be transported to the reference surface and thus be merged: especially trajectories from the main pattern recognition region eventually get lost inside the magnet, for example because the particle was created just there.
For these reasons the analysis routine first splits the *MC tracks* into disjunct sets. Four groups are defined as follows: ① contains particles which are proper tracks in both the Silicon and the Main PR area and② those which aren't proper tracks at all. ③ and④ hold particles that are proper in just one tracking component. The *MC track segment pairs* can then be subdivided into 2×4 groups according to the decision whether both track segments were actually propagated to the reference

surface or not.
It is now possible to define the *merging efficiency* for a single event:

$$\varepsilon_{\mathrm{M}} \stackrel{def}{=} \frac{n_{b,p}}{n_{b,p}^{rec}},$$

where $n_{b,p}$ is the number of segment pairs from ① that were propagated in both trackings, and ^{rec} symbolizes the correctly merged pairs. The *merging extra efficiency* is the fraction of correctly merged pairs from ②, ③, ④:

$$\varepsilon_{\mathrm{M},x} \stackrel{def}{=} \frac{n_{234}}{n_{merge}}.$$

n_{merge} is the total number of merged pairs. The *merging ghost efficiency* is the fraction of tracks incorrectly pasted together:

$$\varepsilon_{\mathrm{M},gh} \stackrel{def}{=} \frac{n_{false}}{n_{merge}}.$$

Mean merging efficiencies are calculated in the same way as for the tracking analysis (see page 17).
The merging analysis also logs the χ^2 and $p(\chi^2, n_{dof})$, the extrapolation residuals $\Delta\vec{p} = \vec{p}_{si}(z_{ref}) - \vec{p}_{main}(z_{ref})$ and the merging errors $\sqrt{\mathcal{C}_{ii}}$. This is done for correctly and incorrectly merged pairs, respectively.

5 Performance

The following two sections summarize the performance of the Silicon pattern recognition and the merging program. The given numbers represent the current status of the program and will hence most likely change as the simulation becomes more detailed and the recognition routines are refined. For this reason the most important digitization parameters and PR algorithm constants are given before we present the actual performance data.

5.1 Pattern Recognition

The simulation uses the following parameters for the conversion from MIMPs to hits (see section 2.):

σ	$12\,\mu$m	coordinate resolution
s_d	$200\,\mu$m	double hit resolution
ε	0.95	detection probability
ε_{noise}	—	noise level

The most important constants for the pattern recognition algorithm are given in the following table. For more detailed information please consult the appendix.

Pattern Recognition Parameters:				
Quantity			Purpose	Reference
Tolerance Window:	m_{min}	$60\,\mu$m	min. size of window	sec. 3.2.2
	m_{max}	$500\,\mu$m	max. size of window	”
	t_m	2 mrad	opening angle of tolerance cone	”
	n_{max}	3	max. no. of accepted prongs in window	”
Track Quality:	f_{miss}	0.2	weight of a fault	sec. 3.2.2
	f_{χ^2}	0.1	weight of χ^2	”
	Q_{min}	8	min. quality of a track prong	”
Misc.:	p_{comm}	0.3	max. allowed degree of equivalence	sec. 3.2.3
	$n_{max,miss}$	2	max. subseq. faults	sec. 3.2.2

Using these numbers with the current program version *SUSi*-1.04/12, 200 events were processed. We obtain the following

Efficiencies:

quantity		*value*	*number of contributing events*	*reference*
ε	$=$	91.2 %	200	sec. 3.3.1
ε_{gh}	$=$	8.5 %	200	"
ε_x	$=$	18.5 %	200	"

Furthermore, it is interesting how well the pions and leptons from the golden decay $B^0 \to J/\Psi\, K_s^0 \to l^+ l^-\, \pi^+ \pi^-$ are recognized. The efficiencies for those particles have been calculated excluding the geometrical acceptance: the J/Ψ for example is regarded as a proper particle if both the π^+ and the π^- are proper. Currently we have:

<table>
<tr><td>ε_{l^+} = 93.0 % (114)</td><td rowspan="2">$\varepsilon_{J/\Psi}$ = 87.1 % (93)</td><td rowspan="4">ε_{B^0} = 60.0 % (15)</td></tr>
<tr><td>ε_{l^-} = 92.0 % (114)</td></tr>
<tr><td>ε_{π^+} = 82.1 % (28)</td><td rowspan="2">$\varepsilon_{K_s^0}$ = 68.4 % (19)</td></tr>
<tr><td>ε_{π^-} = 77.3 % (22)</td></tr>
</table>

quantity = value (no. of contributing events)

Note that $\varepsilon_{l^+}\varepsilon_{l^+} < \varepsilon_{J/\Psi}$, indicating that the probabilties to find a l^+ rsp. a l^+ are not independent. This is e. g. due to kinematical correlations. The same holds for $\varepsilon_{K_s^0}$ and ε_{B^0}.

The recognition probability for a π is smaller than for the leptons: the reason is because a pion typically decays in the middle of the Silicon detector and has therefore less hits. Moreover, the algorithm introduces a slight *malus* for trajectories that stop amidst the sensitive planes. Of course it is necessary to overcome this problem.

Vertex Residual Resolution. Figure 7 shows the distributions of vertex residuals $\Delta\vec{x}$ (see sec. 3.3.2) for all tracks and for the signal decay leptons.

The first one can be described as two superseded normal distributions with $\sigma_1 \simeq 85\,\mu$m, $\sigma_2 \simeq 200\,\mu$m which contribute like $(\mathrm{norm}_1 \div \mathrm{norm}_2) = (4.5 \div 1)$. — Due to lack of statistics the latter was fitted with a single normal distribution, which resulted in $\sigma \simeq 58\,\mu$m.

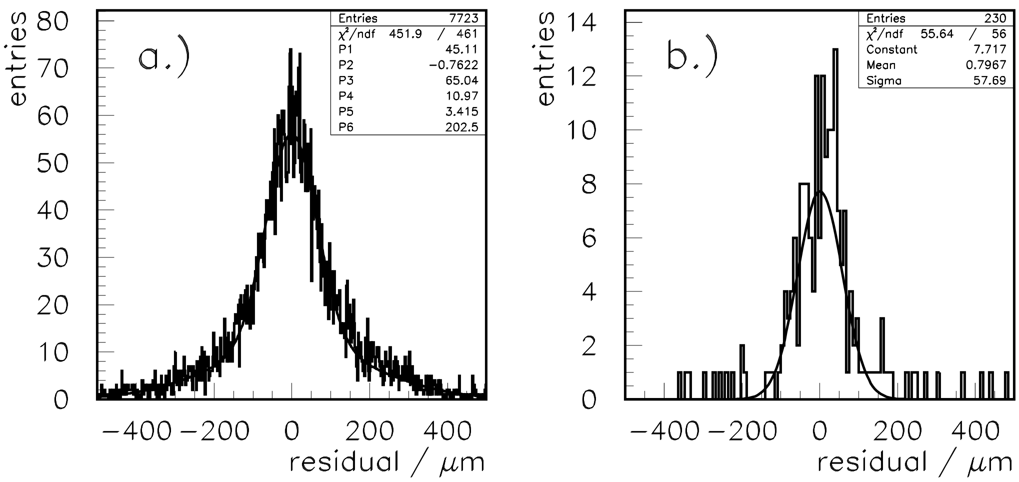

Figure 7: *Vertex residual distribution (vertical component) for:* **a.)** *...all reconstructed tracks. The parameters are from two normal distributions:* $P_{1,4}$ = norm, $P_{2,5} = \bar{x}$ and $P_{3,6} = \sigma$. **b.)** *...the leptons from the signal decay. A single normal distribution has been fitted here.*

Note that **1.)** the vertex $\vec{x}_{Vtx}$ is known more precisely than the vertex residual and that **2.)** the vertex residual resolution can be easily improved by a reasonable outlier rejection strategy.

5.2 Merging

The merging efficiencies have been determined using the same event sample and algorithm parameters as in the previous paragraph. Using the formulas from sec. 4.4 one obtains

quantity		*value*	*number of contributing events*
ε_M	=	95.8 %	200
$\varepsilon_{M,gh}$	=	5.9 %	”
$\varepsilon_{M,x}$	=	—	”
			(Sorting Algorithm)

At the moment no extra efficiency $\varepsilon_{M,x}$ can be given because the ideal pattern recognition in the magnet propagates only those particles to the reference surface

which are proper for the main tracking. Although the subsequent Silicon reconstruction finds considerably more tracks than these, the merging step has thus no chance to associate non–proper segments.
The next–neighbour–algorithm performs equally well. Hence, the two naive approaches with simply a χ^2–cutoff prove to give good results.

6 TBD

Now that a first Silicon pattern recognition algorithm has been written and also the link to the main tracking exists, still a lot of things remain to be done. It's not only that a reconstruction program is never finished but there are some well–defined goals for the next future:

1. Development of a reasonable outlier removal scheme, e.g. by using the Kalman smoother.
2. To benefit from the redundancy produced by both PR algorithms, the track segments that could not be matched in the merging step should be propagated into the rsp. complementary tracking regions.
3. Implementation of vertexing routines.
4. Study of new seeding methods for *SUSi*.
5. "*SUSi–2*", i.e. the development of a local PR algorithm that works in projections.
6. Study of global methods (e.g. [10]). Those are definitely worth a try: even if global methods fail to provide high enough reconstruction efficiencies, they could be used to support the candidate initiation of a local algorithm (see point 4.). This would reduce calculation time considerably.
7. More detailed simulation of the detector modules. This implies the creation of digits in *ARTE* by taking into account: the width of the original charge distribution, diffusion, δ–electrons, interstrip capacitances etc. Of course a reasonable cluster algorithm must be developed for *SUSi* as well.

This (presumably incomplete) list gives an idea of the amount of work that is still to do, and you are invited to join us in the interesting field of pattern recognition.

A Source Code Documentation

A.1 *SUSi*

A.1.1 The Program Hierarchy

Figure 8 shows the routine hierarchy of *SUSi* and how it is embedded in *ARTE*. For information about *ARTE* or *PRISM* please consult [8][6].

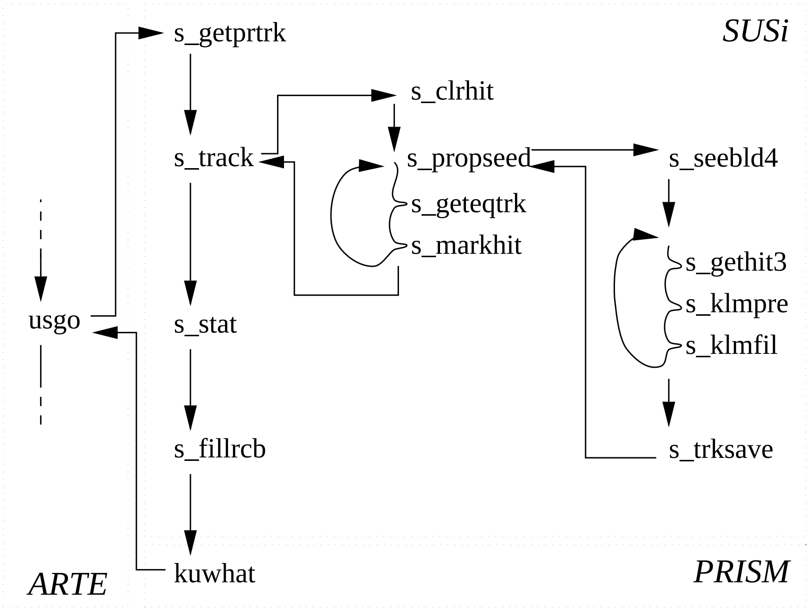

Figure 8: *Simplified program hierarchy and instruction flow in* SUSi. *The program setup is done by the ARTE routine* `usinit`*; it is not shown in the picture.*

A.1.2 `common`–Blocks

This subsection summarizes what the common memory blocks contain and where they are initialized rsp. filled.

`zsiseed` Contains the seeding table. It is initialized by s_geoinit during the program setup. The seeding table gives 1.) a 4–tuple of seeding plane numbers for every pass and seeding index n, and 2.) the total number of seeding indices $n(pass)$.

`zsinavi`	Contains the navigation table. Initilization see `zsiseed`. The navi table gives for every Si-plane 1.) the number of next planes that can be reached and 2.) the plane numbers of these planes.
`zsigeo`	Is a common block containing the Si plane data that must be accessed fast (i.e. without `arte`–pointer fiddling). Initilization see `zsiseed`.
`zsirctrk`	Contains data for both the track candidates and the reconstructed tracks. Candidates filled by `s_trksave`; if a candidate is not accepted as a reconstructed track, then `s_geteqtrk` marks it by setting `rc_ac` to `false`.
`zsiprtrk`	Contains the MC id's of the proper tracks and the rsp. hit id's. Counters for the different selection cuts are also included. `zsiprtrk` is filled by `s_geteqtrk`.
`zsipaw`	PAW common block for the Si–histograms.
`zsihitmark`	Contains a huge `logical`–array that is used to mark the used hits. Filled each event by `s_markhit`.
`zsitrnsm`	Contains the translation matrices $\mathcal{M}^{-1}$ for every seeding combination. Initialized during the program setup by `s_settrnsm`.
`zsigold`	Contains MC–numbers of the golden decay particles. Filled by `s_locb0`.
`zsitrhist`	Contains history for track candidates (needed by smoother algorithm). Filled by `s_trksave` and not by `s_propseed` because operations with `common` variables are at least twice as time–expensive as op's with local declared var's (1.) RTOC–reference necessary 2.) no advantage is taken of the cache.) [11]
`zsiprtrk`	Contains MC–numbers of the proper tracks and counters related to the selection criteria. Filled by `s_getprtrk`.
`zsiconst`	Contains only `parameter`–definitions and is thus a "pseudo" `common` block (`parameter` statements in Fortran are just a macro definition, similar to a `#def` in C). All algorithm parameters are set here on constant values.

A.1.3 Routines

The Silicon pattern recognition routines are described here. The formal parameters are subdivided into `in` parameters (= not changed by the routine but needed as input), `out` parameters (= changed by the routine but not needed as input). `In out` parameters (= changed, but also needed as input) are not given: there are none (or: there should be none `:-)`). However, note that the routines may also depend on variables in `common` blocks which do not appear as formal parameters.

s_geoinit

in : —
out: —

Reads most often used plane data from the *ARTE* table `zgede` and fills them into `zsigeo`. Reads navigation– and seeding tables from files and stores them in `zsinavi` and `zsiseed`.

s_settrnsm

in : —
out: —

Initiliazes the translation matrices $\mathcal{M}^{-1}$ (see 3.2.1).

s_hinit, s_hlast, s_hsave

in : —
out: —

`s_hinit` defines HBOOK ouput file and declares Silicon analysis histograms. Output filename is given "hard–coded" (← change that if you wish). `s_hlast` performs final operations with the histograms (divisions etc.). `s_hsave` saves the histograms.

s_getprtrk

in : —
out: —

Scans the MC particles for proper tracks and logs the cuts: fills `zsiprtrk`.

s_track

in : —
out: —

Steering routine for the Si pattern recognition. Contains 6 nested loops that combine the passes, seeding combinations and hits. Seeds are propagated by calling `s_propseed`. After every pass candidates are selected by `s_geteqtrk` and hits are marked by `s_markhit`.

s_propseed

in : seeding plane id $1-4$, hit id $1-4$, number of pass and seed
out: —

"Heart" of the whole program. Propagates a seeding vector through the detector and collects hits, forks etc. (see 3.2.2).

s_seebld4

in : number of pass and seeding combination, a 4–dim. measurement vector
out: $\vec{p}_0$

Calculates $\vec{p}_0 = \mathcal{M}^{-1}(_{pass,seed}) \cdot \vec{m}$.

s_klmpre

in : $\vec{p}, \mathcal{C}, \Delta z$
out: predicted $\vec{p}, \mathcal{C}$

Fast routine for a Kalman prediction step. The momentum is no `in` parameter, because it is not *a priori* known. The crossed radiation length is assumed to be $\frac{1}{4}X_{superlayer}$.

s_klmfil

in : $\vec{p}, \mathcal{C}, m, \cos(\alpha_{st}), \sin(\alpha_{st}), \sigma_{pl}$

out: filtered $\vec{p}, \mathcal{C}$ and the $\delta\chi^2$

Fast routine for a Kalman filter step.

s_klmsmo

in : $\vec{p}^{\,n}_{k+1}, \mathcal{C}^n_{k+1}, \vec{p}^{\,k}_k, \mathcal{C}^k_k, z_{from}, z_{to}, \sin(\alpha_{st,k}), \cos(\alpha_{st,k}), m_k$

out: smoothed quantities $\vec{p}^{\,n}_k, \mathcal{C}^n_k, r^n_k, \delta\chi^2_{n,k}$

Performs a Kalman smoother step.

s_gethit3

in : $\vec{x}$, id of plane, id of last plane that measured X rsp. Y

out: number of hits in tolerance, arrays with hit numbers and sens. coordinates

Scans the plane for hits that lie in a certain tolerance around $\vec{x}$. The tolerance is calculated from the distance to the last plane. `s_gethit3` delivers at most 3 hits — if it finds more, the closest 3 are taken.

s_gethit3_2

in : $\vec{x}$, id of plane, id of last plane that measured X rsp. Y

out: number of hits in tolerance, arrays with hit numbers and sens. coordinates

Same as `s_gethit3`, but only the closest two hits are taken.

s_markhit

in : total number of candidates so far

out: —

Marks the hits from the reconstructed tracks from a certain pass.

s_clrhit

in : —

out: —

Clears the hit mark array in `zsihitmark` before an event is processed.

s_prntpar

in : $\vec{p}$ and $\mathcal{C}$

out: —

Does what name says: prints $\vec{p}$ and $\mathcal{C}$ nicely formatted.

s_trksave

in : various data that characterize a track candidate

out: —

Saves data from a track candidate in `zsirctrk`. Determines 1st associated MC particle for each of it's hits by calling `tuhmimp`. If there is more than one ass. MC particle, then (MC-id += 10000).

s_geteqtr

in : total number of candidates so far

out: —

Performs the candidate selection (one–by–one comparison) for all candidates from a certain pass. If a candidate is removed, then the logical `rc_ac(cand.-#)` is set to `false`. If preprocessor flag `siprint` is set, the data for the reconstructed tracks are printed.

`s_geteqtr2`

in : total number of candidates so far

out: —

Performs the candidate selection (all–to–all comparison) for all candidates from a certain pass. (Compare `s_geteqtr`.)

`s_pstat`

in : —

out: —

For algorithm diagnosis: analyzes the track candidates, i. e. determines which & how many MC particles are among the track candidates before the candidate selection.

`s_stat`

in : —

out: —

Main analysis routine for the Si pattern recognition (called every event). Determines for each reconstructed track the MC particle that created most of the track's hits and calculates the fraction of these. This results in the classification as "correctly reconstructed" or "ghost". Fills then various evaluation histograms and determines event specific and mean efficiencies for various track groups. Finishes with a great output festival.

`s_getnevt`

in : —

out: number of superimposed events

Determines the number of superimposed interactions in the current event.

`s_getgamma`

in : —

out: —

Searches `zsiprtrk` for γ–daughter pairs and histograms some kinematical parameters. Used to gather knowledge about the γ–cut in s_getprtrk.

`s_locb0`

in : —

out: —

Searches for the golden B–decay and stores MC–numbers of the rsp. particles in `zsigold`.

`s_fillrcb`

in : —

out: —

When all passes have been processed, this routine fills the reconstructed tracks into the *ARTE* tables `ztrac` and `hfit`.

`s_rayback`

in : —

out: —

1st of the 3 "refit" algorithms. Uses a straight line extrapolation to calculate the intersection point at the last plane of measurement. Loops over all recognized tracks.

`s_revfil`

in : —

out: —

2nd of the 3 "refit" algorithms. Performs a Kalman filter go in downstream direction (compare `s_rayback`).

`s_smooth`

in : —

out: —

3rd of the 3 "refit" algorithms. Steering routine for the Kalman smoother: every recognized track is smoothed to it's first plane of measurement.

`s_anaprpg`

in : $\vec{p}, \mathcal{C}, \chi^2$ at the most downstream plane, number of rec. track, algorithm index

out: —

Analyzes the performance of one of the the 3 different refit algorithms.

`s_chiprob`

in : χ^2, n_{dof}

out: $p(\chi^2, n_{dof})$

Calculates the confidence level p for given χ^2 and n_{dof}.

`s_chkseed`

in : —

out: —

Checks if the proper tracks of the current event are caught by a seeding combination. If this is not a case for a certain track, it's data are written to the file `not_seen.dat`.

`s_chkseed2`

in : number of a proper track

out: a `logical` flag

Checks if the proper track given by the `in` parameter is caught by the seeds. If this is the case, the flag is set to `true`, otherwise to `false`.

`s_tstprtrk`

in : —

out: —

Tests which proper tracks would survive the candidate selection step if all of them were recognized perfectly.

`s_getcomm`

in : two numbers of proper tracks

out: fraction of common hits

Calculates the percentage of common hits for two given proper tracks.

`s_chk2miss`

in : number of a proper track

out: a `logical` flag

Checks if the given proper tracks has two faults in sequence. If this is the case, the flag is set to `true`, otherwise to `false`.

s_lppi

in : four MC track id's
out: —

Analyzes various kinematical correlations between the two MC particles 1 and 2 rsp. 3 and 4. As this routine is a pure "add–on", all histograms are booked here at the first call.

s_allks

in : —
out: —

This routine selects all proper pions from all K_s^0 and determines the mean efficiencies for K_s^0 and both π's for: **a.)** K_s^0 that are B^0 daughters, **b.)** K_s^0 that are no B^0 daughters and **c.)** all K_s^0. Informations about every K_s^0 are written to the file `all-ks-pion.dat`.

mygstran

in : *ARTE* plane id
out: —

As indicated by the name, this is not really a *SUSi* routine but it rather belongs to *ARTE*: performs the conversion from MIMPs to measured coordinates and digits for the given plane.

usinit

in : —
out: —

Also an *ARTE* routine. Initializes the geometry that is used in `s_geoinit` and the subsequent `mygstran` calls.

A.1.4 Functions

The function's name and type are given in the first column.

real s_erfc

in : x
out: complementary error function

Calculates $(2/\sqrt{\pi}) \cdot \int_x^\infty e^{-t^2}\, dt$.

reals_trkqual

in : χ^2. number of hits and faults for a candidate
out: Q

Calculates track quality Q.

boolean in_plane_2

in : $x_{min}, x_{max}, y_{min}, y_{max}, \vec{x}$
out: logical flag

Checks if $\vec{x}$ lies in a plane defined by the x's and the y's. Provides that plane is not rotated.

A.2 Merging Algorithms

A.2.1 The Program Hierarchy

Again, the reconstruction is steered by the *ARTE*–routine `usgo` (compare A.1.1). The (simplified) calling sequence is:

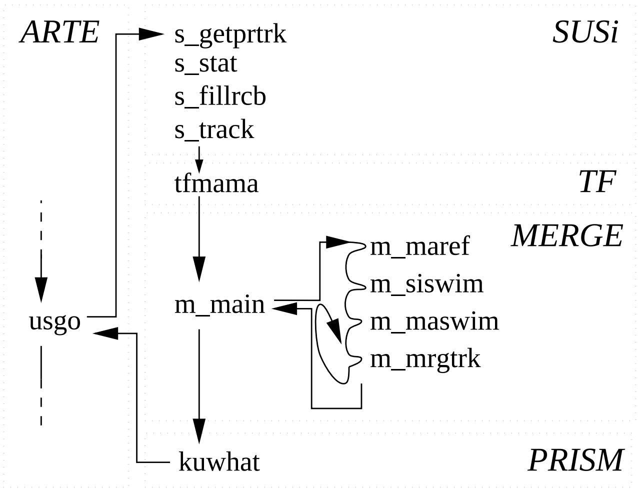

Figure 9: *How the merging package is embedded in ARTE and the main tracking TF. The program setup is again done by the ARTE routine* `usinit`*; it is not shown in the picture.*

A.2.2 `common`–blocks

As the merging package mainly deals with already existing data, there are very few common memory regions.

`zmrcbank` Contains the track parameters of the Silicon and the Main PR track segements at the Silicon exit window. Is filled by `m_siswim` rsp. m_maswim.

`zmmbank` This data structure associates the track segments: it contains the number of merged tracks and for each track pointers into the segment bank `zmrcbank`.

`zmgldlog` Contains logicals that indicate if a "golden" particle is a certain proper track and if it has been found and propagated by the tracking algorithms. Filled every event by `m_fllzgl`.

`zmconst` Corresponding to `zsiconst`, this sequence is no real common block but serves merely to make constants visible.

A.2.3 Routines

Again, we denote the formal parameters of the routines as `in`– rsp. `out`– parameters.

`m_siswim`

in : —

out: —

Loops over the reconstructed tracks of the Silicon and "refits" them to the exit window. The precise distribution of dead matter is taken into account. If preprocessor switch `MOMKNOWN` is set, the MC truth is used for the particle momentum.

`m_maswim`

in : —

out: —

Propagates the proper tracks of the main pattern recognition to the exit window by using `tfidea` and `tfswim`. Rejects a failed backpropagation (∞ or `NaN` in a matrix element).

`m_maref`

in : —

out: —

Finds the proper tracks of the main pattern recognition. Fills main tracking `common`–block `zmmbnk`.

`m_main`

in : —

out: —

Steering routine of the merging step. Calls `m_maref`, propagates the segments with `m_siswim` and `m_maswim`. If preprocessor flag `MESORT` is set, the sorting algorithm is used; if `MESIMPLE` is set, the next–neighbour–alg. is used; if non of the above is set, you get an (intended) compiler error. Finally, `m_main` analyzes the merging result with the help of `zmgldlog`.

`m_hinit`

in : —

out: —

Initializes histograms for the merging package.

`m_geoeff`

in : —

out: —

A routine that should be included in another package: determines the geometrical efficiencies based on the proper tracks for both trackings. Makes excessive use of `zmgldlog`.

`m_fllzgl`

in : —

out: Number of golden decays

Fills `zmgldlog` and returns number of golden B–decays; called every event.

m_klmpre

in : $\vec{p}, \mathcal{C}, \Delta z$, p, ms

out: $\vec{p}_{pre}, \mathcal{C}_{pre}$

Swims track parameters through a given distance Δz. The multiple scattering contribution from the starting Z–position is added which is a function of: ms=(mean ms angle)^2 for a particle with 1 GeV and p=momentum of the particle. `m_klmpre` converts internally to `double`.

m_klmfil

in : $\vec{p}, \mathcal{C}, m, \cos\alpha_{st}, \sin\alpha_{st}, \sigma$

out: $\vec{p}_{fil}, \mathcal{C}_{fil}, \delta\chi^2$

Performs a Kalman filter step. Converts internally to `double`.

m_mrgtrk

in : $\vec{p}_1, \mathcal{C}_1, \vec{p}_2, \mathcal{C}_2$

out: $\vec{p}, \delta\chi^2$, status flag

Merges the two track parameter sets 1 and 2: this results in an "average" $\vec{p}$ and a $\delta\chi^2$. Status flag takes different values if the inversions fail (see doku in source code).

A.2.4 Functions

As in A.1.4, the function name is given as well as the function type.

boolean m_inwall

in : wall–id, vector in XY–plane

out: logical

Checks if vector lies in the specified wall, returns `true` if this is the case. Provides that wall is of *GEANT*–shape `box` and that it is not rotated.

integer m_nSLhits

in : pointer in `zsirctrk`, n

out: number of other hits

The track given by the pointer intersects certain Silicon arms. `m_nSLhits` determines how many other hits are in the arm where the hit n is. This function is needed by `m_siswim` to calculate the correct radiation length per measurement plane.

boolean found_MaTrk

in : MC track number

out: —

Checks if the MC track was found by the main pattern recognition.

boolean found_SiTrk

in : MC track number

out: —

The same as `found_MaTrk`, but for the Silicon.

`integer m_tr2mc`
in : `ZTRAC` track number
out: —
Gives MC track number for a given track from the `ZTRAC` bank.

`integer m_mc2tr`
in : MC track number
out: —
Gives `ZTRAC` track number for a given MC track.

References

[1] The HERA–B collaboration: An Experiment to Study CP Violation in the B System Using an Internal Target at the HERA Proton Ring (Technical Design Report), DESY–PRC 95/01, 1995

[2] R. Frühwirth, Application of Kalman Filtering to Track and Vertex Fitting, NIM A262 444, 1987

[3] H.–J. Hotop: Hochgenaue Flugnavigation durch den Einsatz von Kalman–Filtern, TBD

[4] P. Billoir: Track Element Merging Strategy and Vertex Fitting in Complex Modular Detectors, NIM A241 115, 1985

[5] A. Hüpper: Der ARGUS–Silizium–Detektor und seine Anwendung auf die Bestimmung der Lebensdauer des τ–Leptons, PhD–thesis, Heidelberg 1995

[6] R. Mankel: *prism* — the HERA–B event display, HERA–B internal note 95–011, `http://castor.ifh.de/ subgroup/ software/ welcome.html`, 1995

[7] R. Mankel: Application of the Kalman filter technique in the HERA–B track reconstruction, HERA–B internal note 95–017, `http://castor.ifh.de/ subgroup/ software/ welcome.html`, 1995

[8] S. Nowak: HBGEAN (HeraB Geant) and HBRCAN (15/03/95), HERA–B internal note 94–009, `http://castor.ifh.de/ subgroup/ software/ welcome.html`, 1995

[9] O. Nachtmann: Phänomene und Konzepte in der Elementarteilchenphysik, Vieweg, 1989

[10] M. Ohlsson, C Petersson: Track Finding with Deformable Templates — the Elastic Arm Approach, Computer Physics Comm. 71, 77–98, 1992

[11] N. M. Doerner: Power–Tuning, c't 11–95, 386–390, 1995

Ack's

I would like to thank Hartwig Albrecht and Siegmund Nowak for *ARTE*, the old *HBRCAN* and *HBGEAN*. It is not enough to thank Rainer Mankel only for *PRISM* and the many detailed discussions from which I benefitted a lot: of equal importance are all the beers & pizzas, the slide shows and the debates about cameras and films.

Finally, I would like to thank Peter Kreuzer and Michael Schmelling for proof–reading this document.