SLAC-PUB-6190
May 1993
(T/E)

Deformable Templates—Revisited and Extended —With an OOP Implementation *

Richard Blankenbecler
Stanford Linear Accelerator Center
Stanford University, Stanford, California 94309

In this note, a general approach to track reconstruction will be described, the deformable template or elastic arms algorithm. The review part of the paper is meant to be pedagogical with the physical interpretation of the algorithm kept paramount. The extension of this approach to handle hit ambiguities (mirror charges), delta rays, vee-tracks, etc., will be given. Finally, the implementation of the algorithm in an object oriented computer language will be sketched and the resulting advantages discussed.

I. INTRODUCTION

In this note, a general approach to track reconstruction will be described; it will be a recasting and extension of the work of Ohlsson, Peterson, and Yuille, [1,2] The formalism is applicable to general geometries; for simplicity and clarity of presentation however, I will describe a small solid angle detector, not a full 4π detector as described in these earlier references. To further simplify the discussion, there will be NO magnetic field. The detector arrays are planes in the $x-y$ plane with the predominate beam direction along z . These assumptions are not at all necessary and should not be taken as a limitation of the general method; they are made to ease the burden of description.

This paper is meant to have a strong pedagogical bent. For completeness, the reader should read works using other approaches [3] or the extensive list in references [1] and [2]. [4–10,12] Other works that may prove useful are found in references [4]– [12]. A unified treatment of the original and previous works on this method will be given as well as a few extensions. The discussion contains two distinct topics. The first is a review and extension of the basic mathematical formulation. The second is a discussion of the implementation of this approach using an object oriented programming language .

We shall start by discussing track reconstruction in an ideal detector, add hit coordinate resolution and then add noise hits. The extension of the method to the problem of resolving true ambiguities in the hit coordinates will be described. The inclusion of delta rays, vees that originate inside the active volume and propagate only through a portion of the detector, multiple scattering of tracks in the detector medium, etc., will either be discussed or indicated with varying degrees of completeness. Finally, the extension to a non-zero magnetic field can be made by replacing the word track in the following by trajectory.

II. PERFECTION, NOISE AND AMBIGUITIES

The "deformable template" or "elastic arm" approach to this problem starts by defining an energy function that is minimized when the assumed parametrized tracks fit the given hits; however, some care will be taken to insure that the hits are not *over fit* (that is, arbitrarily introducing enough tracks to pass through every hit). To this end, and following Refs. [1] and [2] very closely, we introduce for a *perfect* detector a total energy function that is linear in the energy of each hit:

$$E_{\text{total}} = \sum_h E[h] \quad , \tag{1}$$

where the energy for a particular hit h , $E[h]$, depends upon all the tracks:

$$E[h] = \sum_t A_{ht} M_{ht} \quad , \tag{2}$$

*Work supported by Department of Energy contract DE–AC03–76SF00515.

where M_{ht} is a suitably chosen measure of the "miss" such as the distance of closest approach (or its square), and A_{ht} is an assignment, or binary decision unit, such that $A_{ht} = 1$ if track t belongs to hit h and is zero otherwise. At the moment, we have not explicitly denoted the dependence of $E[h]$ on the parameters of the track. This simple formulation does not yet contain some very important physical effects present in any real track reconstruction problem.
Resolution: A finite spatial resolution for each hit can be introduced through the definition of the miss distance. For example, one can define the quantity M_{ht} in terms of the ratio of the distance of closest approach of the track to the hit position divided by the spatial resolution of that hit (much in the spirit of a standard chi-squared fit).
Noise: One obvious omission is noise in the system; that is, not all hits are due to the passage of a charged particle. The inclusion of noise is straightforward (see ref [1]). It is implemented by modifying the "assignment" decision variable A_{ht} so that it can also take on the value zero for all t . The hit energy is then written as

$$E[h] \;=\; \sum_t A_{ht} M_{ht} \;+ \nu \left\{ \sum_t A_{ht} - 1 \right\}^2 \;. \tag{3}$$

Thus classifying a hit as noise implies that its energy is ν whose value serves as a regulator for this assignment. Indeed, the parameter ν is a physical parameter — eventually set by the properties of the detector. For a noisy detector, ν is small; for a quiet detector, ν is large. In the general case, ν can depend upon position in the detector as well.
Ambiguities: Ambiguities are treated in exactly the same spirit as above. In the case of a two-fold ambiguity, one introduces the quantities M^+_{ht} and M^-_{ht} which are the miss distance measures between the track t and the two possible locations of the hit h ,

$$E[h] \;=\; \sum_t A_{ht} \left(a^+_{ht}\; M^+_{ht} \;+ a^-_{ht}\; M^-_{ht} \right) + \nu \left\{ \sum_t A_{ht} - 1 \right\}^2 \;, \tag{4}$$

where the new variables a^+_{ht} and a^-_{ht} must satisfy $a^+_{ht} + a^-_{ht} = 1$ and are defined by

$$a^+_{ht}\; (a^-_{ht}\;) = \begin{cases} 1 & \text{if hit } h^+(h^-) \text{ is assigned to track } t \\ 0 & \text{otherwise .} \end{cases} \tag{5}$$

The interpretation of the assignment variables is now that A_{ht} assigns the hit-pair either to noise or to the track t , whereas the $a^\pm_{ht}$ determines which member of the ambiguity pair is assigned to t .
Tasks: At this point there are several problems ahead of us. The first is estimate the number of tracks (in a later section we shall extend our class "track" to include delta rays, vees, kinks, etc.). An initial estimate of their (location) parameters is also needed to initialize the fitting procedure. These now form a set of elastic arms that must be fit to the data. Finally, a minimization procedure must be devised that minimizes the total energy $E_{\rm total}$ by choosing both the assignment variables and the track parameters, yet allows some of the hits to be assigned to noise. Let us now turn to this latter problem.

III. SIMULATED ANNEALING

Our mathematical problem is to find the global minimum of the energy function assuming that the number of tracks and estimates for their parameters are all known. An efficient method for treating this problem is simulated annealing (see ref [1]). All of the following machinations are to develop a robust yet simple fitting procedure. We shall find, however, that there are many unexpected bonuses; the procedure yields physical interpretations that have real meanings and are of considerable utility. The first step is to introduce a Boltzmann distribution for the assignment variables and the track parameters

$$P[\,\mathbf{A}\,;\,\mathbf{p}\,] = \frac{1}{Z}\; e^{-\beta E[\,\mathbf{A}\,;\,\mathbf{p}\,]} \;, \tag{6}$$

where $\mathbf{A}$ $(= \{A_{ht}\})$ is the set of assignment variables, $\mathbf{p}$ $(= \{p_t\})$ is the set of track parameters, and β is the inverse temperature. Finally, the partition (normalization) function is

$$Z = \sum_A \sum_p e^{-\beta E[\,\mathbf{A}\,;\,\mathbf{p}\,]} \tag{7}$$

in which the sum goes over *all* allowed values of $\mathbf{A}$ and $\mathbf{p}$

Now let us compute the *marginal* probability that describes the distribution of track parameters for a *uniform* distribution of assignments $\mathbf{A}$

$$P[\,\mathbf{p}\,] = \sum_{A_{ht}} P[A_{ht};\,\mathbf{p}\,] \;\equiv\; \frac{1}{Z}\, e^{-\beta E_{\rm eff}\;[\,\mathbf{p}\,]}\;, \tag{8}$$

where an *effective* energy has been defined that will prove convenient in later manipulations and clearly

$$E_{\rm eff}\;[\,\mathbf{p}\,] = \sum_{h}\; E_{\rm eff}\;[h,\,\mathbf{p}\,]\;. \tag{9}$$

The evaluation of the sum over the allowed values of $\mathbf{A}$ is straightforward. For a fixed value of h , the element $A_{ht} = 1$ if h is assigned to t and thus $\mathbf{A}_{\,hu} = 0$ if $u \neq t$. The hit is assigned sequentially to each separate value of t or it is assigned to zero, $A_{ht} = 0$ for all t, if the hit h is assigned to noise; the energy function is then M_{ht} or ν , respectively. The marginal probability without ambiguity is then

$$P[\,\mathbf{p}\,] \;=\; \frac{1}{Z}\prod_{h} D(h) \tag{10}$$

$$\text{where} \qquad D(h) \;=\; \left\{ e^{-\beta\nu} + \sum_{t} e^{-\beta M_{ht}} \;\right\}\;, \tag{11}$$

and the effective energy of the hit h is

$$E_{\rm eff}\;[h,\,\mathbf{p}\,] = -\frac{1}{\beta}\log D(h)\;. \tag{12}$$

If hit ambiguities are included, then one must sum over the two possibilities for each set $a^{\pm}_{ht}$. The result is the simple replacement in $D(h)$, recall equation ,

$$e^{-\beta M_{ht}} \;\;\longrightarrow\;\; \frac{1}{2}\;\left(e^{-\beta M^{+}_{ht}} \;+ e^{-\beta M^{-}_{ht}}\;\right)\;. \tag{13}$$

Finally, note that each track t may have several parameters. To clearly denote this, the notation change $p_t \;\rightarrow \vec{p}_t$ will be adopted.

Minimization: We are looking for the most probable configurations, i.e. the number of tracks, their parameter values and the values of the assignment variables that minimize the energy in the limit of low temperatures. In this limit of large β , the wrong assignment configurations, i.e. those with a finite miss distance M_{ht} , are exponentially suppressed in the marginal probability. Hence in order to find the best fit to the hit data, we choose to minimize the effective energy $E_{\rm eff}\;[\,\mathbf{p}\,]$ and follow the track parameters for a range of increasing values of β in order to avoid being trapped in a local minima.

Using the gradient descent method, at each stage in the iteration the parameters of each track t are changed by

$$\delta\vec{p}_t \;=\; -\eta\;\overrightarrow{\bigtriangledown}_{p} E_{\rm eff}\;[\,\mathbf{p}\,] \;=\; -\eta\;\sum_{h}\;\overrightarrow{\bigtriangledown}_{p} E_{\rm eff}\;[h,\,\mathbf{p}\,]\;. \tag{14}$$

The scalar parameter η is used to control the rate of approach to the minimum. See ref [2] if p is a mixture of linear, angles, etc., in which case a tensor η is required.

Interpretation: An explicit evaluation of the derivatives leads to a simple but useful interpretation of this process and of the quantities involved in the calculation. Note that the derivative of the energy $E[h]$ with respect to M_{ht} is the assignment A_{ht}. The derivative of the effective energy with respect to the same variable is

$$\frac{\partial\, E_{\rm eff}\;[h,\,\mathbf{p}\,]}{\partial\, M_{ht}} \;=\; \frac{e^{-\beta M_{ht}}}{D(h)}\;. \tag{15}$$

This suggests that one introduce the thermalized assignment probability of hit h to track t as

$$< A_{ht} > = \;\frac{e^{-\beta M_{ht}}}{D(h)}\;, \tag{16}$$

and the probability of assigning hit h to noise as

$$< \text{noise}(h) > = \frac{e^{-\beta\nu}}{D(h)} \ , \tag{17}$$

with the normalization

$$\sum_t \ < A_{ht} > \ + \ < \text{noise}(h) > = \ 1 \ . \tag{18}$$

This simply states that a hit must be assigned either to one of the tracks or to noise. The gradient descent equations can now be written as

$$\delta\vec{p}_t \ = \ -\eta \ \sum_h < A_{ht} > \ \overrightarrow{\bigtriangledown}_p M_{ht} \ \ . \tag{19}$$

Thus it is seen that hits with a large value of the assignment probability, $< A_{ht} >$ dominate the determination of the track parameters. This also shows that reasonably accurate initial estimates of the track parameters will be needed at the start of the fitting procedure.

When ambiguities are present, one also has the quantities

$$< a^+_{ht} \ > = \ \frac{e^{-\beta M^+_{ht}}}{e^{-\beta M^+_{ht}} \ + e^{-\beta M^-_{ht}}} \quad \text{and} \quad < a^-_{ht} \ > = \ 1 - \ < a^+_{ht} \ > \ . \tag{20}$$

In this case the gradient descent equations take the form

$$\delta\vec{p}_t \ = \ -\eta \ \sum_h < A_{ht} > \ \left\{ < a^+_{ht} \ > \ \overrightarrow{\bigtriangledown}_p M^+_{ht} \ + \ < a^-_{ht} \ > \ \overrightarrow{\bigtriangledown}_p M^-_{ht} \right\} \ , \tag{21}$$

where

$$< A_{ht} > = \ \frac{1}{2\,D(h)} \ \left(e^{-\beta M^+_{ht}} \ + e^{-\beta M^-_{ht}} \right) \ . \tag{22}$$

The simulated annealing technique is to follow the solutions for the parameters $\mathbf{p}_t$ at or at least near the minimum in the energy function as β is increased; at the lower values, the assignment of the hit-pair to a track is made and when $\beta * (M^+_{ht} \ - M^-_{ht} \ ) \gtrsim 1$, the ambiguity starts to be resolved.

Other interesting quantities are the rough measures:

$$N[h] \ = \ \sum_t \ e^{-\beta M_{ht}} \ \sim \ \text{number of tracks that pass thru hit } h, \tag{23}$$

$$N[t] \ = \ \sum_h \ e^{-\beta M_{ht}} \qquad \sim \ \text{number of hits along track } t, \tag{24}$$

which are meaningful after the fit has converged sufficiently. Indeed, unless there are true track crossings, $N[h]$ is 1 or 0 when the algorithm has converged.

Bayesian Interpretation: In Ref [1] it was pointed out that maximizing $P[\,\mathbf{p}\,]$ can be thought of as performing a maximum likelihood estimation of the parameters $\mathbf{p}$ of a model that generates hit data $\mathbf{H}$ with probability

$$P[\,\mathbf{H}\,,\,\mathbf{p}\,] = \frac{1}{Z_H} \prod_h \left\{ e^{-\beta\nu} + \sum_t e^{-\beta M_{ht}} \ \right\} \ . \tag{25}$$

We then apply Bayes' theorem to obtain

$$P[\,\mathbf{p} \ : \ \mathbf{H}\,] = \frac{P[\,\mathbf{H} \ : \ \mathbf{p}\,] \ P[\,\mathbf{p}\,]}{P[\,\mathbf{H}\,]} \ . \tag{26}$$

If we make the reasonable assumption that the prior probability of $\mathbf{p}$, $P[\,\mathbf{p}\,]$ is uniform, then finding the best *aposteriori* estimate, maximizing $P[\,\mathbf{p} \ : \ \mathbf{H}\,]$ with respect to $\mathbf{p}$, is equivalent to maximizing $P[\,\mathbf{H} \ : \ \mathbf{p}\,]$.

The β-parameter is now interpreted as a measure of the spread of the distribution and must be estimated from the detector characteristics and the definition of the miss distance, M_{ht} . Our probability distribution therefore assumes that the data comes either from one of the templates or from a distribution parameterized by ν.

IV. CLASSIFYING AND INITIALIZING OBJECTS

So far the discussion has mentioned only tracks as the mathematical constructs to be fit to the data. There are several topologies that must be dealt with. These include tracks, delta rays, vees, kinks, etc.

Tracks are defined as simple curves (straight lines in the present discussion) which pass through the detector. Delta rays are low energy tracks that originate in the detector volume due to the passage of a high energy particle and produce a "swarm" of localized hits. No attempt will be made to fit these hits in detail; a rough localization and classification will be assumed sufficient. Vees are a charged pair that arises inside the detector from the decay of a neutral parent, and propagates on through the volume. To include decays or scattering from elements of the detector, one introduces a "Kink" class that is composed of several straight line tracks (much as is the Vee class) but with the tracks joined in sequence. Note that Vees and Kinks are constructed out of track segments; they are therefore carriers of their constituent track objects.

These are rather incomplete definitions, but when this approach is realized in object oriented programming language code, they will provide the physical basis for defining classes, their instance variables and their methods.

Alternative Strategies: There are several different approaches that can be used to implement the fitting procedure. Roughly these can be described as recursive, global and mixed. The *recursive* strategy tries to find the preferred type of fitting object and to estimate its parameters, say, from the most significant peak in the Hough transform (see next paragraph). It then starts with a small value of the noise parameter ν and tries to fit this one object to the data. If this succeeds, it then increases ν and repeats the process on the remaining hits which at this point are classified as noise. This procedure is repeated until the fit is not improved by adding a new track or until it is reasonable to treat the remaining hits as noise. This scheme will allow one to fit an event in which there are two closely spaced and almost parallel tracks, for example. The *global* strategy tries to find a complete set of objects that would exhaust the fit to the data, again from the Hough transform. All the objects are then fit to the data simultaneously. The *mixed* strategy gets an initial set of objects from the most significant signal in the transform and fits these first before repeating the process on the remaining hits.

The Hough Transform: This brings us to the next problem, that of estimating the number of objects to be fitted to the hits, their type (i.e. track, delta ray, etc.) and rough estimates for their parameters. This task is conventionally carried out by means of the "Hough" transform, or in other words, histogramming.

It is most convenient to search for the "swarm" classes first. To this end, a histogram is generated for the total number of hits on each of the detector planes; this will be termed a z-histogram. If the count for a particular plane exceeds a threshold value, then it is classified as a delta ray plane and is further analyzed for the rough location and extent of the swarm. The threshold should depend upon the average number of hits per plane for the event under study. After the size of the swarm is roughly determined, then all hits lying inside this swarm area are tentatively declared as delta ray hits; they can then be removed from consideration during the next few stages of the Hough analysis.

The remaining hits in the z-histogram are then examined, looking for values of z where the number of hits in the planes jumps by two. If any such point is found, a possible vee is declared and its z-value, denoted by z_{vee} , is noted. Footnote: The point of origin of the vee is of course in front of the first jump point z_{vee} .

Following Refs [1] and [2] , the remaining hits are then paired (there are several possible strategies to limit the number of pairs that must be examined) and fits made to extract track fitting parameters. These are then histogrammed. Two histograms are generated, one for hit-pairs with $z < z_{vee}$, and the other for $z > z_{vee}$.

Tracks that pass through the detector should show up as peaks in both histograms. The two tracks in the vee should show up in the latter and not the former; a differential comparison of these histograms will yield estimates of the parameters of the vee objects, including its vertex position.

V. REFORMULATION

Now that several new classes of objects have been introduced, it is perhaps useful to rewrite the general treatment to explicitly denote this fact and to extend the terminology. The object of the fitting exercise is to assign independent hits to a track, a delta ray, or a vee. That is, the hits are to be *grouped* into an object with a physical meaning and a simple parametrization or to random noise. Instead of the sum just over tracks, the energy is written as a sum over group types g :

$$E[h] \;=\; \sum_g A_{hg} M_{hg} \;+\nu \left\{ \sum_g A_{hg} - 1 \right\}^2 \quad . \tag{27}$$

More physically, the sum g is over tracks t , delta rays d , and vees v :

$$E[h] = \tag{28}$$

$$\sum_t A_{ht} M_{ht} + \sum_d A_{hd} M_{hd} + \sum_v A_{hv} M_{hv} \tag{29}$$

$$+ \nu \left\{ \sum_t A_{ht} + \sum_d A_{hd} + \sum_v A_{hv} - 1 \right\}^2 . \tag{30}$$

A hit must be classified as belonging to a track, a delta ray, a vee, a kink, or to noise. The miss distance variables M_{ht} and M_{hv} are familiar variables measuring the distance of closest approach (see Appendix A for examples; also recall that a vee does *not* extend completely through the detector). However the probability of occurrence of a delta ray depends on the detector itself so it is a physical parameter. For a hit on a detector plane and inside the area that has been designated as a delta ray region, the miss parameter will be denoted by δ ; for a hit outside the region or off the plane, the miss parameter is a large constant, say infinite, which forces the corresponding assignment probability to be zero.

Thus a hit classified as noise will cost an energy ν , whereas a hit classified as a delta ray will cost δ . Note that during the fitting process, if a track happens to pass through the delta ray region and the miss parameter M_{ht} for the hit h is *less* than δ , then as the fit converges this hit will automatically be reclassified by the assignment variable $< A_{ht} >$ as a track hit. Thus δ should be chosen larger than the miss distance corresponding to the hit resolution, yet less than the miss distance corresponding to the average hit separation in a delta ray swarm.

At this stage we have not imposed the requirement that a delta ray be associated with a nearby track. This requirement could be added to the analysis by, for example, adding a penalty term to the energy that vanishes if a track passes sufficiently near a delta ray region. A *very simple* example of the procedure is given in Appendix B.

The Kink class is composed of several tracks (similar to the Vee class) but which are joined in sequence and extending through the detector. The constraint on the fit is that each track pair join at a point; the zmax of the leading track is equal to the zmin of the following track. Note that at a mature stage of the annealing process, one may replace a track instance by a kink instance to see if the fit can be improved significantly. Such modifications are straightforward to implement using an object oriented approach .

Finally, hit ambiguities can be included in the discussion by replacing each miss distance by

$$M_{hg} \longrightarrow \left(a^+_{hg} \, M^+_{hg} + a^-_{hg} \, M^-_{hg} \right) . \tag{31}$$

Note that for a delta ray assignment, $M^+_{hd} = M^-_{hd} = \delta$, the $a^{\pm}_{hd}$ sum to one, and there is no ambiguity to resolve.

VI. IMPLEMENTATION IN OOP

In this section, an implementation of the elastic arms approach to track reconstruction will be described using object oriented programming techniques . The language C++ will be used for definiteness, but the important point is the use of classes and their instances (objects) to ease the task of programming. The resultant code will have a simple and physical structure, be easy to maintain and to modify. The use of an object oriented programming language will considerably simplify the treatment of this type of problem in which the number of hits, tracks, delta rays, etc. vary from event to event and are not known a priori. Indeed, they may change during the fitting of an event.

In this discussion a simplified problem will be treated. It will be assumed that each detector plane yields information about one direction only, x or y or a linear combination (the wire plane may rotated through an angle ϕ). There is also an ambiguity in position related to the pulse time delay; the hit may be on either side of the wire or rather a given radial distance from the wire.

Some of the design decisions involved here are quite straightforward. Those quantities that describe a particular hit and involve a sum over group objects will be placed in the Hit class. Quantities that are functions of a particular group object, a track say and involve a sum over all the hits, belong obviously in the Group class and its subclasses.

The hits are the basic physical input data; this information is stored in an object of the Hit class which contains instance variables giving its label, location, and certain parameters utilized in the fitting/annealing process that satisfy the above criteria.

The design of the other classes is based on the following observations. Only the classes descended from Group, i.e. Track, DeltaRay, Vee, Kink, etc., know what their parameters are and indeed, how many there are. Thus each member of this set of classes must know how to vary its full set of parameters so as to minimize the total energy. The gradient descent method, as given in equation 19, does require that each group object perform a sum over all hits in

order to update its parameter values. Thus the overall control program need not know the details of the class, only that each class will respond properly to the commands calcMiss(), calcAssign(), and varyParams(). Each class will respond to these messages in its own way; this use of polymorphism considerably simplifies writing the fitting code as well as the job of adding new group classes.

In order to respond, each group class needs access to the hit data, i.e. the hit locations, and the parameters of the annealing process; it should also contain arrays of the miss distances and their derivatives with respect to the group parameters. The assignment array could be placed in either the Hit or the Group class; it is located in the Group class to reduce messaging during the fitting process.

The Hit class is defined as:

```
class Hit {
 public:
        Hit( );
       ~Hit( );
  void  setZHdHCos(float ,float ,float ,float )
/* Public utility print & set/get methods go here.*/
  private:
   int  label;
   float z, h, dh, hcos,resolution;
   float (*hSag) (..);
   float Eeff, D, N, noise;
   BOOL  owned;
   int   owner;
  }
```

where the meaning of label and z are clear.

The height of the hit is h (wire number?) and dh is the ambiguity distance. The instance variable hcos is the cosine of the wire angle in the detector planes, such that h=x for hcos=1, and h=y for hcos=0; resolution is also obvious. The function pointer *hSag(..) corrects h for any sag in the wire and its arguments are the transverse coordinates of the track. Each wire plane will have its own sag function so this pointer is set at initialization of the data. All the following variables change during the fitting process; they also depend upon temperature. The quantity Eeff, given by equation 12, is the energy of this hit; D is defined in equation 11 and involves a sum over group objects and their miss distance. N is the expected number of group objects that pass through this particular hit, equation 24 , and the probability of this hit being classified as noise, as given by equation 17, is so labeled. The owned and owner variables allow one to track the assignment of this hit to a group object.

Finally note that the Hit class is simply a data repository. Therefore, it could just as well be defined as struct Hit. This may be more efficient for batch mode running, especially from anexternal (FORTRAN) driver.

The abstract Group class is defined by the header file:

```
class Group {
 public:
           Group( );
  virtual ~Group( );
  virtual  void  setLabelGroup(int ,enum groupType );
  virtual  void  initialize(void) { };
  virtual  void    calcMiss(void) = 0;
  virtual  void  calcAssign(void) = 0;
  virtual  void  varyParams(void) = 0;
/* Many public utility print & set/get methods go here.*/
 protected:
  int    label;
  enum   groupType grp;
  static Hit **hit;
  static int numHits;
  static float beta, lambda, eta;
  float  zmin, zmax;
  float  missp[MAXHITS], missm[MAXHITS];
  float  A[MAXHITS], ap[MAXHITS], am[MAXHITS];
  float  N;
```

```
}
```

where label is for bookkeeping convenience, and the enum variable groupType is { group, track, deltaRay, vee, kink} . The static variables are used by all the members of the class Group (and its subclasses) during the fitting process. All members of the Group class extend over a finite extent from zmin to zmax; hits outside this range are given a miss distance of infinity. Now each group object needs to know its miss distance from every hit position [13]; missp is the miss from (h+dh), and missm from (h-dh). The array A[h] is the assignment probability of hit h to this object, and the arrays ap[ ] and am[ ] are the relative probabilities for the hit positions (h ± dh). N is the expected number of hits that this particular group object fits, see equation 24 . The code that precisely defines the operations of the above methods are contained in the implementation file for the Group class. In writing this code, one may want to add other methods, usually private or protected and accessible only to the class itself, to aid in structuring the code design.

The subclasses of the Group class will inherit its protected instance variables and virtual methods. As the first example, the Track class is given by:

```
class Track : public Group {
 public:
            Track( );
   virtual ~Track( );
   virtual void initialize(void);
   virtual void calcMiss(void);
   virtual void calcAssign(void);
   virtual void varyParams(void);
   virtual void setX0X1(float ,float );
   virtual void setY0Y1(float ,float );
/* Public utility print & set/get methods go here.*/
 protected:
  float x0, x1;
  float y0, y1;
  float dMisspdh0[MAXHITS], dMisspdh1[MAXHITS];
  float dMissmdh0[MAXHITS], dMissmdh1[MAXHITS];
  void  calcNewX0X1(void);
  void  calcNewY0Y1(void);
 }
```

Each subclass must implement the pure virtual functions calcMiss(), calcAssign(), and varyParams(). Each will also add its own set of parameters and the methods needed to search for the best fit. The track is parametrized by the forms x = x0 + (x1-x0)∗z and y = y0 + (y1-y0)∗z; since the hit h is given by a tilted wire plane, one has h = x∗hcos + y∗hsin. The arrays dMisspdh0[h] and dMisspdh1[h] are the derivatives of missp with respect to the two orthogonal track parameters. These arrays are calculated and loaded in the calcMiss() method.

The DeltaRay class header file contains:

```
class DeltaRay : public Group {
 public:
            DeltaRay( );
   virtual ~DeltaRay( );
   virtual void initialize(void);
   virtual void calcMiss(void);
   virtual void calcAssign(void);
   virtual void varyParams(void);
/* Public utility print & set/get methods go here.*/
 protected:
  float z;
  float hmax, hmin;
 }
```

The variable z gives the central plane in which the delta ray occurs; the delta ray extends from zmin to zmax. The variables hmax and hmin measure the transverse spatial extent of the delta ray. These can be considered to be parameters to be varied, or as fixed numbers given by the Hough transform (see Appendix B for an example). If they

are fixed, then the method varyParams() is a null method; it does nothing. For hits that fit the criteria for delta rays, the DeltaRay class assigns them a miss of δ .

The Vee class demonstrates the use of object oriented programming in building up a complex class from simpler elements; the two tracks that make up the Vee are instances of the Track class. Since they each know how to vary their parameters during the fitting procedure, the methods of the Vee class make use the methods already present in the Track class. The arrays in the base Group class are used to record the final results; namely the best value of A[h] and its corresponding miss[h]. that are present in the two track instances, the p(ositive) and n(egative) Tracks. The coordinates of the apex, zv,xv and yv, are calculated from the intersection point of the two tracks and recorded in the Vee object. The Vee class header file contains:

```
class Vee : public Group {
 public:
           Vee( );
   virtual ~Vee( );
   virtual void calcMiss(void);
   ......................
   virtual void setFitParamsPM(float ,float ,float ,float );
 protected:
  Track *pTrack;
  Track *nTrack;
  float zv;                  // z-value of apex of Vee.
  float xv, yv;              // x- and y-value of apex.
  virtual void calcZvXvYv(void);
 }
```

The constructor and a sample method are of the form:

```
   void  Vee::Vee() {
              pTrack = new Track;
              nTrack = new Track;
              zv = xv = yv = 0.0;
              pTrack->setZmin(zv);
              nTrack->setZmin(zv);
        }
   void  Vee::calcMiss(void) {
              pTrack->calcMiss();
              nTrack->calcMiss();
        }
```

Following the above example of a class that *contains* other (simpler) classes, the Kink class is to contain a number of connected straight line track segments; the Kink class header file contains:

```
class Vee : public Group {
 public:
           Kink( );
   virtual ~Kink( );
   virtual void calcMiss(void);
   ......................
 protected:
  int numTracks;
  Track **track;
  float *zv;                  // array of apex z-values.
  float *xv, *yv;              // array of apex x- and y-values.
 }
```

The constructor and a sample method are of the form:

```
   void  Kink::Kink() {
         for (i=0; i<numTracks;i++) {
                        track[i] = new Track;
```

```
                zv[i] = xv[i] = yv[i] = 0.0;
              track[i]->setZmin(zv[i]);
          }
       }
  void  Kink::calcMiss(void) {
       for (i=0; i<numTracks;i++) track[i]->calcMiss();
       }
```

The Controller and Classifier classes are somewhat arbitrary but are chosen to allow easy maintenance and modification. For example, the main controller class, MControl, simply initializes the program by creating an instance of the Control class, reading the input data files, and instructing control to set up the hit list. Control then creates an instance of the Hough class to examine the hit data for an estimate of the number, type, and parameters of the Group classes such as Track, DeltaRay, Vee, etc., and creates instances of these classes in response to the findings of Hough. Control then creates an instance of its friend class Anneal to carry out the simulated annealing procedure which adjusts the parameters in the Group objects to minimize the energy.

Thus if one wants to develop alternative algorithms for fitting, only the Anneal class needs to be modified. In a similar way, the Hough class localizes the histogramming/estimation methods. Finally, the Display class localizes the dependence upon the GUI chosen, and much of this class can be written to be portable.

In the Anneal class, the temperature is initialized and then the list of Group objects are ordered to vary their parameters so as to minimize the energy. This proceeds by first messaging the group list to calculate their miss variables (the calculations are carried out in the subclasses since they know the fitting parameters, and the appropriate derivatives are computed here as well). Then the hit list is told to calculate their variables (which involves sum over the just computed group values). Next, the group is instructed to compute their assignment variables, $A[h]$, $ap[h]$ and $am[h]$, and finally told to vary their fit parameters. The temperature is gradually lowered and the above sequence repeated until convergence is achieved.

A sample header for the Control class could contain:

```
class Control {
 public:
        Control( );
       ~Control( );
  friend class Hough;
  friend class Anneal;
  void start(void);
  void   loadHitObject(int ,float ,float );
  void   loadTrackObject(float ,float );
  void   loadDeltaRayObject(float ,float ,float );
  void   loadVeeObject(float ,float ,float );
/* Many public utility print */
/* & set/get methods go here.*/
 private:
  MControl *mcontrol;
  Hough *hough;
  Anneal *anneal;
  Hit *hit[MAXHITS];
  int numHits;
  Group *group[MAXGROUP];
  int numGroup;
  int numTracks;
  int numDeltaRays;
  int numVees;
  float nu;      // The annealing parameters start here....
  float delta;
  float betaStart;
  float delBeta;
  float beta;
  ......................
  void setGroupStatic(void);
  ......................
```

```
}
```

where most of the variables have already been defined. The Control class is responsible for keeping track of all the Hit objects and the Group objects. It therefore has all the parameters needed in the annealing process. In its start() method, the Hough class is called, the appropriate objects are instantiated, and the Anneal object proceeds with the fitting using a thermalized gradient descent method.

The header for the Hough class contains:

```
class Hough {
 public:
        Hough( );
       ~Hough( );
  void   histogramHits(void);
  void   searchAndMakeGroupObjects(void);
  void   lookForTracks(void);
  void   lookForDeltaRays(void);
  void   lookForVees(void);
/* Many public utility print */
/* & set/get methods go here.*/
 private:
  Control *control;
  float aveZDensity;
  int   row;            // Histogram arrays and matrices.
  int   col;
  ..................
  void createNewTrack(float ,float );
  ..................
 }
```

It is in the lookFor methods that the different search algorithms can be implemented. The createNewTrack() method messages the Control instance to add a new track to its Group list and to initialize its parameters.

The Anneal class is also quite simple:

```
class Anneal {
 public:
        Anneal( );
       ~Anneal( );
  void   startThermalDescent(void);
/* Some public utility print */
/* & set/get methods go here.*/
 private:
  Control *control;
  void   doGradientDescent(void);
  ..................
 }
```

It is in the doGradientDescent() method or in alternatives to this method that new minimization schemes can be invoked.

VII. CONCLUSIONS

The deformable template method has a very interesting mathematical structure with a clear physical interpretation. One of its most useful concepts are the neuron-type variable, denoted here by A_{ht} , that "assigns" a track to a hit and ends up being interpretable as the probability that hit h belongs to track t.

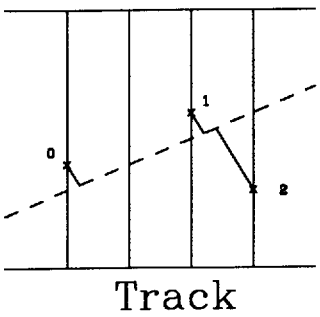
The other useful feature of this algorithm is that it lends itself to encoding using object oriented programming techniques in a very natural way. The use of a object oriented programming language simplifies programming and its physical interpretation as well as the maintenance and extension of the code. As an example, the a Group subclass can be introduced that contains tracks propagating in a magnetic field; this subclass needs to have methods to propagate

itself in the given field and to adjust its parameters to fit the data. Finally, composite classes are introduced, recall the Vee and Kink classes, that have mixtures of field free straightline propagation and curved tracks. The composite classes are simple to implement since their constituents already contain most of the necessary methods.

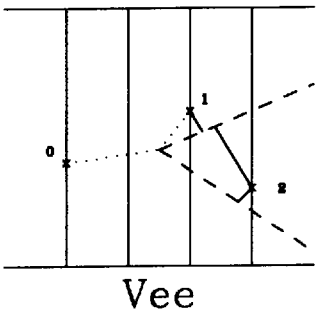
The final test of this approach, of course, requires the development of a robust and fully implemented computer program and its application to real data.

APPENDIX A

In this appendix the choices and definitions for the miss distances for the Group classes will be illustrated. In each figure there are three numbered hits marked with an x. The left figure contains a track (dashed) and the right figure contains a vee (also dashed).



Track



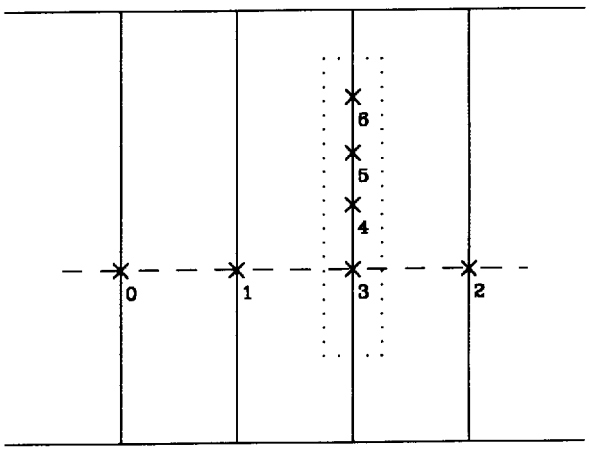
Vee

The distance of closest approach for a track is clear and is illustrated as the solid line. For a vee, there are two miss distances for each hit—the distance of closest approach to each arm of the vee. However, when this point lies to the right of the apex of the vee, the distance to the apex is adopted as a measure of the miss and denoted by a dotted line. Both distances from point 0 are to the apex; these allow the hits to "pull" or "push" the apex of the vee to improve the fit.

APPENDIX B

The procedure will be illustrated by a very simple toy example. Recall that the total energy is the sum of the energies of each hit h which in turn is given in terms of the function $D(h)$, where

$$E_{\rm eff}[\,\mathbf{p}\,] = -\frac{1}{\beta}\sum_h \log D(h) \quad \text{and} \quad D(h) \;=\; \left\{ e^{-\beta\nu} + \sum_g e^{-\beta M_{hg}} \right\} \,. \tag{32}$$



Hits (x), Tracks (dash) and Delta Rays(dots).

It will be assumed that there is no noise, i.e. $\nu \rightarrow \infty$ and that the hits are distributed on the four detector planes as illustrated in the figure. Clearly there is a delta ray swarm on the third plane and a track. By ignoring the swarm in the Hough transform, the remaining hits will suggest initial values for the track parameters; for these latter hits $D(h) = e^{-\beta M_{ht}}$, h=0,1,2 while $D(h) = e^{-\beta M_{ht}} + e^{-\beta\delta}$, h = 3,4,5,6 for the delta ray swarm hits. The parameters defining the track are residing in the miss distance M_{ht} . These parameters are determined from the minimum of the energy function

$$E_{\rm eff}[\,\mathbf{p}\,] = \sum_{h<3} M_{ht} \; - \frac{1}{\beta} \sum_{h>=3} log(e^{-\beta M_{ht}} + e^{-\beta\delta}) \; . \tag{33}$$

The calculation of the minimum with respect to the track parameters searches for zeroes of the gradients

$$\overrightarrow{\bigtriangledown}_T E_{\rm eff}[\,\mathbf{p}\,] \; = \; \sum_{h<3} \overrightarrow{\bigtriangledown}_T M_{ht} \; - \sum_{h>=3} < A_{ht} > \overrightarrow{\bigtriangledown}_T M_{ht} \quad , \tag{34}$$

where

$$< A_{ht} > = \; \left(1 + e^{-\beta(\delta \; - M_{ht} \;)}\right)^{-1} \quad . \tag{35}$$

As β increases, the only term in the second sum that has a chance to survive is for $h = 3$. For this term, M_{ht} can become smaller than δ and $< A_{3t} > \sim 1$, while all the other $< A_{ht} >$'s become small since their miss, M_{ht} , is larger than δ . Thus δ should be chosen to be smaller than the expected miss for delta rays from other delta ray hits, but larger than the resolution of the measured hits. Note that in this formulation there are no parameters in the DeltaRay class that need to be varied.

ACKNOWLEDGEMENTS

Many conversations with Carsten Peterson and Mattias Ohlsson about the elastic arms algorithm are gratefully acknowledged. The author also wishes to acknowledge the many discussions of experimental reality with George Irwin, but to note that he is not responsible for any lack of it in this paper.

---

[1] M. Ohlsson, C. Peterson, A. Yuille, "Track Finding with Deformable Templates—The Elastic Arms Approach," *Computer Physics Communications* **71**, 77 (1992).

[2] M. Ohlsson, "Extensions and Explorations of the Elastic Arms Algorithm," Lund University Preprint LU-TP-92-28. To be published in Computer Physics Communications.
[3] For example, M. Gyulassy and H. Harlander, "Elastic Tracking and Neural Network Algorithms for Complex Pattern Recognition," *Computer Physics Communications* **66**, 31 (1991).
[4] C. Peterson and B. Söderberg, "A New Method for Maping Optimization Problems onto Neural Networks," *International Journal of Neural Systems* **1**, 3 (1989).
[5] C. Peterson, "Parallel Distributed Approaches to Combinatorial Optimization Problems—Benchmark Studies on TSP", *Neural Computation* **2**, 261 (1990).
[6] B. Denby, "Neural Networks and Cellular Automata in Experimental High Energy Physics," *Computer Physics Communications* **49**, 429 (1988).
[7] C. Peterson, "Track Finding with Neural Networks," *Nuclear Instruments and Methods* **A279**, 537 (1989).
[8] G. Stimpfl-Abele and L. Garrido, "Fast Track Finding with Neural Nets," *Computer Physics Communication* **64**, 46 (1991).
[9] L. Gislén, C. Peterson and B. Söderberg, "Rotor Neurons—Basic Formalism and Dynamics," *Neural Computation* **4**, 737 (1992).
[10] C. Peterson, "Neural Networks and High Energy Physics," *Proceedings of the International Workshop on Software Engineering, Artificial Intelligence and Expert Systems for High Energy and Nuclear Physics*, Lyon Villeurbanne, France, March 1990, eds. D. Perret-Gallix and W. Woijcik, Editions du CRNS (Paris 1990).
[11] A. L. Yuille, "Generalized Deformable Models, Statistical Physics, and Matching Problems," *Neural Computation* **2**, 1 (1990).
[12] A. Yuille, K. Honda and C. Peterson, "Particle Tracking by Deformable Templates," *Proceedings of 1991 IEEE INNS International Joint Conference on Neural Networks*, Vol. 1, pp. 7–12, Seattle, WA (July 1991).
[13] It is not necessary to compute all of these distances, if one coordinate difference exceeds a set value, then the miss distance is assigned the value infinity.