The BaBar Track Fitting Algorithm

David N. Brown[1], *Eric A. Charles*[2], *Douglas A. Roberts*[3]

[1] Lawrence Berkeley National Laboratory, USA
[2] University of Wisconsin, USA
[3] University of Maryland, USA

Abstract

We present an update on the Kalman filter track fitting algorithm used by BaBar, which was first presented at CHEP 97 [1]. The novel formulation of the Kalman filter algorithm and its C++ implementation are reviewed. We describe our experience commissioning and running this track fitter on the data produced in the BaBar commissioning run (with around 1.5 fb^{-1} of integrated luminosity and over 100 million tracks recorded). We present preliminary results on its performance. We also describe extensions to this algorithm by which it is now used as a pattern recognition tool and a detector alignment tool in BaBar.

Keywords: Kalman,tracking,pattern recognition,alignment

1 Introduction

BaBar [2] is an HEP detector operating at the PEP-II assymetric e^+e^- B factory collider [3]. The BaBar detector tracking system consists of an inner Silicon Vertex Tracker (SVT) with 340 double sided silicon strip detectors arranged in 5 layers surrounded by a 40 layer axial plus stereo wire drift chamber (DCH). The physics goals of BaBar require precise, accurate track parameters both near the e^+e^- annihilation point (to define the decay positions of short-lived particles) and outside the outer wire chamber (to assist the particle identification devices which surround the tracking volume).

The Kalman filter is well established as the standard formalism for track fitting in High Energy Physics experiments [4, 5]. The BaBar implementation of this algorithm uses a novel formulation of the processing equations which is well suited for Object Oriented programming. This formulation produces optimal track parameters along the full particle trajectory, for each of the five stable charged particle mass hypotheses. The BaBar fitter accounts for the effects of material interactions and magnetic field distortions, which are computed according to a detailed model of the tracking environment. The fitter provides tools for evaluating the consistency of new hits with an existing track fit, which is used to build efficient and sophisticated pattern recognition algorithms. The fitter provides extensions for constraining the fit to external information, which is used in computing both the internal and relative alignment of the BaBar tracking devices. The BaBar fitter and its extensions are run as part of the standard BaBar reconstruction chain.

2 Mathematical Formalism

Track fitting involves determining a parameter vector P and its covariance matrix C used in a parametric function $\vec{F}(P : l)$ describing the local trajectory of a charged particle through space in terms of a suitable 1-dimensional variable l. A typical choice for $\vec{F}(P : l)$ for detectors with a solenoidal magnetic field is a five-parameter helix about the principle field direction, as a function of the transverse distance along the helix. The parameter definition used by BaBar is given in appendix A. For a given track, the fit should find parameter values which optimize the consistency of the trajectory function with the associated measurements (hits), with a covariance expressing the precision of those hits and the dispersion introduced by the tracking environment. A Kalman track fit allows an optimal least-squares fit including effects like scattering in the material interspersed between the measurements without introducing those effects as explicit parameters in the fit.

The BaBar formulation of the Kalman filter track fit is a hybrid of the standard *weighted means* and *gain matrix* Kalman filter formulations [5], in which the processes defined as 'filtering' and 'smoothing' are combined. This results in symmetric equations for processing the track in either direction, which simplifies the fit implementation and corresponds to BaBar's need of accurate parameters at both ends of a track. This formulation lends itself well to Object Oriented programming.

In the BaBar fit, a set of reference parameters R is used both to seed the processing equations and to define a *reference trajectory* $\vec{F}(R:l)$ used to estimate physical effects such as the amount of expected scattering and the hit residuals. These effects are processed sequentially according to their flight length from the track origin (*outwards*), and then in reverse (*inwards*), updating the estimate of the track parameters *for that processing direction* at each step. In the following sections, the specific equations describing how the effects of materials, magnetic field inhomogeneity, and hits on the track are processed in the fit are presented.

After processing in both directions, the optimal fit parameters anywhere along the track can be computed as the statistical combination of the local values of the inwards and outwards fits (see section 2.3 for details). These optimal parameters are used to create a representation of the track as a piecewise helix (helix segments joined sequentially in space). Physical properties of the track such as the particle's momentum and its trajectory through space are accessed using this piecewise helix representation in BaBar physics analysis.

2.1 Material Effects

As a particle traverses matter, it is subject to both energy loss and directional scattering. The effect of material traversal in the BaBar track fit is considered as a discrete change in the parameters and their covariance, subject to the physics model described in appendix B. Interactions with extended materials (such as the gas volume in a tracking chamber) are modeled as a series of discrete interactions. Scattering and energy loss variance are treated as 'process noise', while mean energy loss is treated during parameter 'transport'.

Equation 1 describes how the BaBar fit models the change in parameters P and covariance C for the passage of a particle of momentum p and mass m through a thickness Δx of material M. Θ and Φ refer to the two independent scattering angles, and Ψ to the fractional momentum change (due to particle energy loss). These variables are defined precisely in appendix B, as are the derivatives $\frac{\delta P}{\delta\Theta}$, $\frac{\delta P}{\delta\Phi}$, and $\frac{\delta P}{\delta\Psi}$. Formulae for $\frac{dE}{dx}$ (mean energy loss), $\sigma^2_{\Delta E}$ (energy loss variance), and σ^2_S (scattering angle variance) for different materials and particles are taken from [8] and [9]. As can be seen, the equations are linear, prompting the definition of material parameters and covariance P_M and C_M.

$$\begin{aligned} P' &= P + \begin{cases} +P_M & \text{outwards} \\ -P_M & \text{inwards} \end{cases} , P_M \equiv \frac{\delta P}{\delta\Psi}\cdot\frac{\sqrt{p^2+m^2}}{p^2}\cdot\frac{dE}{dx}\Delta x \\ C' &= C + C_M \, , C_M \equiv \left(\frac{\delta P}{\delta\Theta}^T\frac{\delta P}{\delta\Theta} + \frac{\delta P}{\delta\Phi}^T\frac{\delta P}{\delta\Phi}\right)\sigma^2_S + \frac{\delta P}{\delta\Psi}^T\frac{\delta P}{\delta\Psi}\frac{p^4}{p^2+m^2}\sigma^2_{\Delta E} \end{aligned} \quad (1)$$

2.2 Magnetic Field Inhomogeneities

The parameterization choice for a track generally intrinsically describes the deflection of a charged particle in the nominal magnetic field present in the tracking environment. However, deviations of the field from the nominal value (inhomogeneities) deflect the particles from the idealized trajectory function $\vec{F}(P:l)$. The BaBar fit provides a way of correcting the trajectory for magnetic field inhomogeneities a special transport of the track parameters. Since the Kalman formalism requires discrete transport, the continuous effect of the field inhomogeneities is modeled as a series of discrete effects, analogous to the treatment of continuous material.

The field inhomogeneity effects are computed using a detailed magnetic field map, derived from field survey data. The initial trajectory is divided up into many, small pieces in regions of large inhomogeneities, and a few, large pieces in regions of constant, nominal field. The momentum change induced by the field inhomogeneity is computed for each piece as the path integral $\vec{\Delta p} = \int_{l_{min}}^{l_{max}} (\vec{B}(\vec{F}(R:l)) - \vec{B}_{\text{nom.}}) \times \vec{dl}$. This momentum change induces a change in the track direction, which is modeled by a change in track parameters at the midpoint of the integrated piece according to the following equation:

$$\begin{aligned} P' &= P + \begin{cases} +P_B & \text{outwards} \\ -P_B & \text{inwards} \end{cases} , \; P_B \equiv \frac{\delta P}{\delta\Theta}\frac{\vec{\Delta p}}{\mid \vec{p} \mid} \cdot \hat{\Theta} + \frac{\delta P}{\delta\Phi}\frac{\vec{\Delta p}}{\mid \vec{p} \mid} \cdot \hat{\Phi} \\ C' &= C \end{aligned} \tag{2}$$

The angles Θ and Φ are defined as for material effects (see appendix B). The uncertainty in the field map is small enough that its propagation to the track covariance matrix is not necessary.

2.3 Hits

The effect of a measurement on the fit is expressed by first calculating the residual r of a tracking hit to the reference trajectory. An estimate of the hit weight ($w \equiv 1/\sigma^2$) and the derivative of the residual with respect to the reference parameters $L \equiv \frac{\delta r}{\delta P} \mid_R$ are required. In BaBar, the residuals are computed as the distance of closest approach in space between the hit and the reference trajectory, and the residual derivatives are computed numerically.

A simple set of equations for hit processing can be derived in *weight-space*, defined in terms of the tracking parameters and covariance in equation 3. The inverse relations $C = \gamma^{-1}$ and $P = C\beta$ follow trivially. Equation 4 defines related quantities to describe a hit in weight space[1].

$$\gamma \equiv C^{-1} \quad , \quad \beta \equiv \gamma P \tag{3}$$

$$\gamma_H \equiv L^T w^2 L \quad , \quad \beta_H \equiv L^T w(LR - r) \tag{4}$$

From these definitions and the Kalman filtering equations, a simple linear equation can be derived for the effect of adding a hit to a track:

$$\beta' = \beta + \beta_H \quad , \quad \gamma' = \gamma + \gamma_H \tag{5}$$

The BaBar fit processes hits interspersed with material by transforming between parameter and weight space as necessary, so that the processing equations are always expressed as simple addition. Note that, in this formulation, no matrix inversion is required to process adjacent hits, reducing the total number of matrix inversions relative to other formulations of the Kalman fit [4, 5]. Note also that the reference parameters are hidden in the definition of β_H, making it trivial

[1]Our definition of β_H corresponds to a residual signed as measurement - prediction

to combine hits which use different reference parameters. This in turn facilitates updating the hit derivatives during iteration.

Weight space is also convenient for computing the optimal parameters after a full inwards and outwards processing. From least-squares first principles, it is possible to derive the following equations for the optimal track parameters at any position l along the track:

$$\beta_{opt}(l) = \beta_{out}(l) + \beta_{in}(l) \quad , \quad \gamma_{opt}(l) = \gamma_{out}(l) + \gamma_{in}(l) \tag{6}$$

The term $\beta_{out}(l)$ represents the value of β after processing outwards up to position l on the reference trajectory. The other terms in equation 6 are defined analogously. The optimal parameters and their covariance can be computed from $\beta(l)$ and $\gamma(l)$ according to the trivial inverse of equation 3.

3 The BaBar Track Fit Code

The BaBar track fit is coded in C++ according to an object-oriented design driven by the underlying mathematics. It is integrated with the general BaBar tracking framework [7]. The fit is written generally enough to accommodate any reasonable track parameterization.

3.1 Basic Processing Classes

The basic processing equations of the BaBar track fit are embodied in the classes `KalParams` and `KalWeight`. `KalParams` contains a `HepVector` and a `HepSymMatrix` [2] to represent the parameter vector and its associated covariance matrix respectively. A `KalParams` object may physically represent an intermediate or final fit result, or the effect of a piece of material (see section 2.1) or a magnetic field inhomogeneity integral (see section 2.2). Parameter processing for material or magnetic effects is provided by `KalParams` operators += and -=, which act by adding covariance matricies and adding (subtracting) parameter vectors, for outwards (inwards) processing, respectively.

The `KalWeight` class contains a `HepVector` and a `HepSymMatrix`, representing the weight vector β and weight matrix γ of equation 3. A `KalWeight` object may physically represent the intermediate result of a hit processing, the effect of a particular hit, or the intermediate result of merging inwards and outwards weights to compute optimal results. Weight-space processing is implemented by `KalWeight` operator +=, which simply adds its constituents.

Classes `KalParams` and `KalParams` implement cross constructors and cross-equivalence operators according to equation 3, to facilitate the transformation of the fit information content between parameter and weight space. Because the cross-construction of a `KalParams` object from a `KalWeight` object (or converse) involves a matrix inversion which may fail, both classes contain a flag indicating their validity, which is set to false in the event of a matrix inversion error. All `KalParams` and `KalWeight` functions which use matrix operations are protected by first checking this validity. This is necessary to insure robustness in the fit.

3.2 Kalman Sites

The virtual base class used to describe any effect on the track (material, field inhomogeneity, hit, or other) is `KalSite`. `KalSite` contains two `KalParams` objects and two `KalWeight` objects, one for each processing direction (inwards and outwards), setup as lazy-evaluated caches. Accessor functions to the `KalParams` (`KalWeight`) cache automatically cross-construct the cache if not

[2] `HepVector` (n-dimensional vector) and `HepSymMatrix` (n-dimensional symmetric matrix) are classes defined in the `CLHEP` class library

already present and the equivalent `KalWeight` (`KalParams`) cache is. The `KalSite` constructor takes as input the reference trajectory, which is used to compute any physical effect (such as a hit residual) and any derivatives. The flight length along the reference trajectory at which this effect occurs is computed by the subclass constructor and stored as a data member, to allow sorting `KalSite` objects.

`KalSite` processing is defined by the pure virtual 'process' function, which takes as input the previous `KalSite` object and the processing direction. Processing requires the previous `KalSite` to have a valid cache (either `KalParams` or `KalWeight`) in the given direction. After being processed, the `KalParams` or `KalWeight` cache (as appropriate) for that direction becomes valid, allowing it to be used for subsequent processing. The paragraphs below define how 'process' is implemented for specific `KalSite` subclasses.

The ability to iterate the BaBar Kalman track fit is provided by the pure virtual function 'update'. This takes as input a revised estimate of the reference trajectory, and is implemented in subclasses to recompute the physical effect for this new reference. The `KalParams` and `KalWeight` caches in both directions are invalidated by 'update'.

The concrete `KalSite` subclass `KalMaterial` describes the effects of a piece of material along the track. It contains a `KalParams` data member representing the material effect, computed as P_M and C_M of equation 1. `KalMaterial` implements the 'process' function in parameter space, adding (subtracting) its `KalParams` object to the one coming from the input `KalSite` for outward (inward) processing, respectively.

The concrete `KalSite` subclass `KalBend` describes the effects of a magnetic field inhomogeneity integral. It contains a `KalParams` data member representing the field inhomogeneity integral, computed as P_B (and 0 for the covariance matrix) of equation 2. `KalBend` implements the 'process' function in parameter space, adding (subtracting) its `KalParams` object to the one coming from the input `KalSite` for outward (inward) processing, repectively.

The concrete `KalSite` subclass `KalHit` describes the effects of a hit on the track. It contains a `KalWeight` data member representing the hit, computed as β_H and γ_H of equation 4. `KalHit` implements the 'process' function in weight space, adding its `KalWeight` object to the one coming from the input `KalSite`, for both inwards and outwards processings.

4 The Kalman Fit Track Representation

The BaBar Kalman track fit is organized by the class `KalRep`. Objects of this class represent individual fits of a track; for instance, the different mass hypothesis fits of a track are separate `KalRep` objects. `KalRep` contains all the objects and functions necessary to perform the fit, and contains the result of that fit.

The `KalRep` constructor takes as input the list of hits to fit as well as the reference parameters (presumably the result of a simpler fit of these same hits). The constructor builds the reference trajectory from the reference parameters, and builds `KalSite` objects; `KalHit` objects are constructed for each hit, `KalMaterial` objects are constructed for each piece of material found to be intersected by the reference trajectory, and `KalBend` objects are constructed for each section of an appropriate division of the reference trajectory. These `KalSite` objects are stored in a vector sorted by their flight length parameter.

The BaBar track fit is performed by the `KalRep` 'fit' function. This processes the `KalSite` vector in order, passing the previous `KalSite` object to the 'process' function of each successive object. The first `KalRep` `KalSite` is processed by passing it an artificial `KalSite` object constructed from the reference trajectory parameters with a covariance matrix inflated by a factor of 10^6. The `KalSite` list is then processed in reverse order. Optimal parameters are then computed for every non-hit site [3] according to equation 6, and a piecewise helix object is generated from that set of parameters. Fit convergence is tested by comparing this trajectory with the reference trajectory: If the difference exceeds a preset tolerance, the piecewise helix just computed becomes the new reference trajectory, the elements of the `KalSite` list are reset by calling 'update' (with the new reference trajectory), and the chain of outwards and inwards processing is repeated.

5 Performance of the BaBar Track Fit

The algorithm described in this paper has been the primary track fit in BaBar since its completion two years ago. During 1999 it has been used to reconstruct the data obtained in the BaBar commissioning run, where around 1.5 pb^-1 of $\upsilon(4s)$ data were recorded, with over 10^8 tracks reconstructed. After initial tuning and bug fixing, the track fit has performed well. Initial reliability problems involving crashes due to failed matrix inversion were solved by the protections described in section 3.1, and aborting the fit if inversion failed. The impact parameter resolution observed with this fit is shown in figure 1.

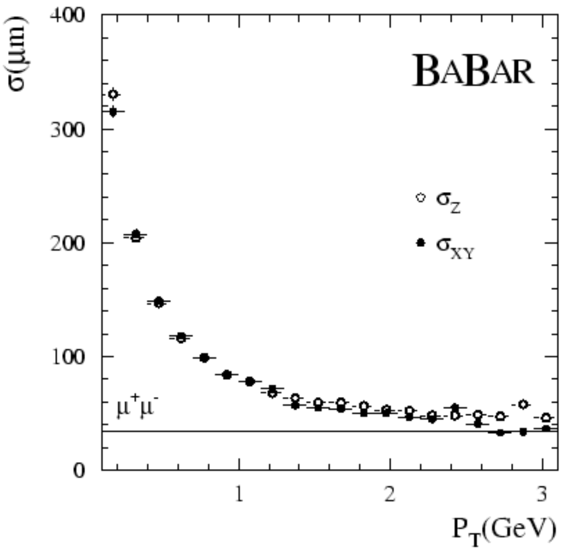

Figure 1: Impact parameter resolution for BaBar tracks as a function of their transverse momentum

The fit (including the pattern recognition extension described in section 6.2) takes roughly 10% of the total cpu time required to fully reconstruct a BaBar event. The cpu time in the fit was found to divide roughly evenly into 5 catagories: computing hit residuals, processing the fit, computing material interactions, constructing the piecewise helix trajectory, and computing magnetic field integrals. Over half the time to run the fit was spent in the `KalRep` constructor, in the construction of the material, hit, and field, `KalSites`. The time in those constructors was dominated by geometric calculations (finding materials intersected by the reference trajectory, computing residuals, and computing integrals). The time to construct the piecewise helix is mostly spent finding the endpoints of the component helix segments. These endpoints are not predicted by the fit equations, and a geometric minimization is required to determine them accurately.

[3]Using equation 6 it can be shown that the optimal parameters do not change between hits.

While discontinuities in the piecewise helix are not formally allowed by the fit equations, second-order effects do cause small discontinuities. These are generally very small (a few microns), but can be substantial (visible in an event display) when a hit is incorrectly assigned to a track. The parameter precision is correctly estimated even when discontinuities are visible.

6 Pattern Recognition with the BaBar Fit

As other authors have noted the Kalman filter can also be used as a powerful pattern recognition tool [5]. In particular, the Kalman implementation provides an efficient mechanism for computing the consistency of a candidate (new) hit with an existing track. Since the Kalman formalism includes scattering and other physical effects not easily treated in simpler track models, it can provide superior accuracy when assigning hits to tracks.

The disadvantage of using a Kalman filter track fit as a pattern recognition tool is the high expense of computing this consistency compared to similair measures in simpler models. In particular, to evaluate the consistency of a second hit after adding a hit to a Kalman track generally requires an expensive reprocessing of large segments of the existing track.

BaBar has developed a pattern recognition algorithm which avoids this problem in two ways. First, it provides an inexpensive algorithm to select reasonable candidate hits before using the Kalman fit consistency check. Second, it only permits adding hits to an end (either inward or outward) of a track. This allows making optimal consistency measures for a sequence of hits, including the effect of newly-added hits, without needing an extensive update of the track.

Restricting new hits to one end of a track constrains the sophistication of the pattern recognition algorithms that can be used. In particular, it precludes testing different sequences of candidate hits as a group. The BaBar pattern recognition algorithm avoids this constraint by considering a tree of possible extensions of a track, with branches for each candidate hit. By appropriate branching and pruning, a truely optimal solution can be found.

The BaBar Kalman filter pattern recognition algorithm is divided into two pieces. The first is defined by class `KalStub`, which provides a specialized interface to an existing `KalRep` object that supports testing candidates and adding hits to one of its ends. The second is defined by class `TrkHitAdd` [6], which provides candidate hit selection and branch manipulation functionality. These two pieces are described in the following sections.

6.1 `KalStub`: a Pattern Recognition Tool

The `KalStub` class serves as an interface between Kalman fitting and pattern recognition. As such, its main function is to compute the consistency that a particular candidate hit came from a given track.

Each `KalStub` is constructed from a (const) `KalRep` 4 object, and the direction (inwards or outwards) in which new hits are to be added. The input `KalRep` need not have been fully fit when passed to `KalStub`: only the specified direction need have been processed for it to be useable. Fitting the `KalRep` in only one direction avoids wasting time on processing which would be invalidated by the addition of hits during the track extension. The partial `KalRep` fit is seeded by a helix fit to track previously found by stand alone track finding algorithms, in only part of the detector. `KalStub` does not alter the `KalRep` to which it points, either on construction or when used.

On construction of a `KalStub` the reference trajectory of the `KalRep` is swum through that part of the tracking environment not covered by the initial fit, and `KalMaterial` and `KalBend` sites are constructed and stored on the `KalStub` for later use. Processing of these sites is deferred to avoid wasted computations should hits be added during track extension.

The KalStub function 'chisq' measures the consistency of a candidate hit as the residual χ^2, using a linear approximation to update the residual for the effect of new hits already added to the KalStub. The linear approximation avoids the (costly) recalculation of the residual with respect to the new fit parameters. The *corrected residual* r_L and the residual χ^2 calculation used by KalStub are presented in equation 7.

$$\begin{aligned} r_L &= r + L^T \cdot (P - R) \\ \chi^2 &= r_L \cdot (w^2 + L^T C L) \cdot r_L^T \end{aligned} \tag{7}$$

The two terms inside the parenthesis in equation 7 describe the covariance contributions from the hit and the error on the orbit of the track respectively. Before computing the residual χ^2, KalStub verifies that all its KalSites up to the flight length of the added hit have been processed, and processes them if they have not. This insures the accuracy of the terms in equation 7.

Because the residual χ^2 is relatively expensive to compute, KalStub does not throw away the result of function 'chisq'; instead, it keeps the test as a candidate KalHit object in cache. Thus several candidate hits may be tested, and the optimal chosen after comparing and selecting the most consistent. When the best hit is selected, it is permanently associated with the KalStub by invoking the 'add' function. Function 'add' copies the cache KalHit object for that hit into the permanent part of the KalStub, and clears the rest of the cache.

Crucial to their use as a pattern recognition tool KalStub can be cloned, to allow the user to follow multiple hypothesis branches during pattern recognition. A cloned KalStub references the same KalRep as the original, and has a (deep) copy of the original's cache, allowing any hit from the cache to be chosen without any redundent processing. Branch development and testing is described in sections 6.2.

Once pattern recognition is completed, a KalStub can be reintegrated with the KalRep object upon which it is based. This reintegration transfers all the new KalSites found and processed during the extension and hit adding of the KalStub to the KalRep. To complete the fit of this extended KalRep requires only fitting in the opposite direction from the one used to add hits, as all the original and added KalSites have valid caches in the other direction. This recycles the processing done during pattern recognition in the fitting, avoiding any redundent processing.

6.2 Use of KalStub in Pattern Recognition

Although the KalStub functionality described above provides a near optimal interface for computing hit consistency, full pattern recognition requires additional functions such as candidate pre-selection, branch generation and pruning, and across-track hit arbitration. These functions are provided by the class TrkHitAdd.

To avoid wasting time computing χ^2 on the thousands of noise hits generated by machine background each event, a pre-selection of hit candidates is required. In TrkHitAdd this selection is based on the detector geometry. Two interrelated classes, TrkHitCombos and TrkDetElements, are used for this. TrkHitCombos serves as the basic candidate building blocks in the TrkHitAdd software; they represent self-consistent combinations of hits from a TrkDetElement, a small region of the detector. Since both the geometrical representation and physical interpretation of hits differs greatly between the SVT and the DCH, separate concrete implementations of TrkDetElements and TrkHitCombos are required for both sub-systems.

The TrkDetElements are used to greatly narrow the search conditions, by swimming each are track through a geometrical model of the detector and determining which TrkDetElements are near enough to a particular track for their associated hits to warrant consideration for inclusion on that track. Furthermore, sorting the TrkDetElements in order along the track assures that the associated hits are processed in the correct order by the pattern recognition.

The TrkHitCombo is to provide an encapsulation of a hypothesis as to which hits from a particular TrkDetElement to associate with a track. By defining a measure of quality of fit:

$$Q_{combo} = \Sigma_i^{hits} \chi_i^2 + P(miss, geom, time) \tag{8}$$

where P is a penalty assigned for non-χ^2 effecs such as missing hits, inconsistent timing, and geometrical consistency of the set of hits. This penalty technique also allows combinations including missing hits or even no hits at all to be treated identically other types of combinations.

In the TrkHitAdd branching is encapsulated as class TrkBranch, which manages and provides access to an associated KalStub. The TrkBranches implement hypothesis proliferation by allowing the user to generate a child TrkBranch by adding a single TrkHitCombo onto an existing TrkBranch. This operation handles cloning and updating the underlying KalStubs as well as storing the parent-child relationships between the TrkBranches.

The branch proliferation algorithm proceeds by iterating over TrkDetElements in the direction of track extension. For each TrkDetElement the existing set of TrkBranches is used in conjunction with the TrkCombos associated with that element to create a set of child TrkBranches which include consideration hit from the detector regions associated with the TrkDetElement in question.

To avoid having an exponentially large set of TrkBranches, every step of iteration includes a pruning process in which the TrkBranches are sorted by quality and all but a small subset of the most favored are discarded. Using the sum of the quality measures from the TrkHitCombos $Q_{branch} = \Sigma_i^{combo} Q_i$ provides an excellent measure of branch quality for this purpose with essentially no computational overhead.

Once the processing has progressed all the way through the extension region, one TrkBranch is selected for each track as the most likely to provide the best measurement of that track. The selection criteria is a simple extension of the branch quality defined above.

$$Q_{branch'} = Q_{branch} + P(pattern, time, dedx) \tag{9}$$

Where P is an arbitrary penalty assigned to branches which have unlikely pattern of hits, hit times or ionization information. This penalty is crucial to the handling of missing and fake hits; without considering the branch as a whole this algorithm presents no way to recognize the not infrequent cases when a fake hit happens to lie along the trajectory of the track in question, but is for other reasons obviously not for that track. The most obvious example of such a case is the association of hits before the starting point of tracks in the physical processes $K_s^0 \rightarrow \pi^+\pi^-$ decay and photon conversion. Under conditions of high detector occupancy, it is common for a few spurious hits from beam related background tracks to lie near the extrapolation of a track.

In the most general case, the decision of which hits to associate with which track can depend on whether that same hit is also being considered for inclusion on another track. Although this problem is much less severe at a comparatively low multiplicity and jet-free environment like BaBar than at higher energy or hadronic colliders, physical processes like $K_s^0 \rightarrow \pi^+\pi^-$ and photon conversion are particularly susceptible to hit confusion.

Since removing a hit from a Kalman fit has the effect of invalidating processing of subsequent hits it is computationally advantageous to perform hit arbitration during the branch selection process. In that way, if a particular hit is present on the most favored branch of two tracks it possible to select instead a slightly less favored branch on which the hit in question is not present for one of the tracks and avoid the need to do any reprocessing of hits.

Once a TrkBranch has been selected for each seed track and all of the processed KalSites contained on that TrkBranch, are integrated back into the seed track's Kalman fit, a fully processed fit exists in the direction of extension. To complete the fitting process, each fit is then processed in the opposite direction and iterated if necessary. The fact that all of the fitting operations are performed by the `KalSite` classes regardless of whether the Kalman fit was perform as part of the pattern recognition or independently insures that the fit results are consistent independant of the context in which the fitting was done.

7 Performance of Track Extension in BaBar

In BaBar `TrkHitAdd` is used to associate SVT hits to tracks that have already been found in the Drift Chamber (DCH). This method of pattern recognition has demonstrated a higher efficiency and superior resistance to backgrounds compared to separately performing stand alone SVT track finding and subsequently matching SVT and DCH tracks. The stand along SVT track finding is now used primarily to find those tracks with too little transverse momentum to be reconstructed in the DCH. Figure 2 plots the efficiency for associating SVT hits to a track using the `TrkHitAdd` algorithm, compared to a stand alone SVT pattern recognition algorithm based on requiring four space points to define a track. In some regions of the detector with damaged or broken silicon modules, the `TrkHitAdd` algorithm improves the efficiency from .60 to .98 as seen in

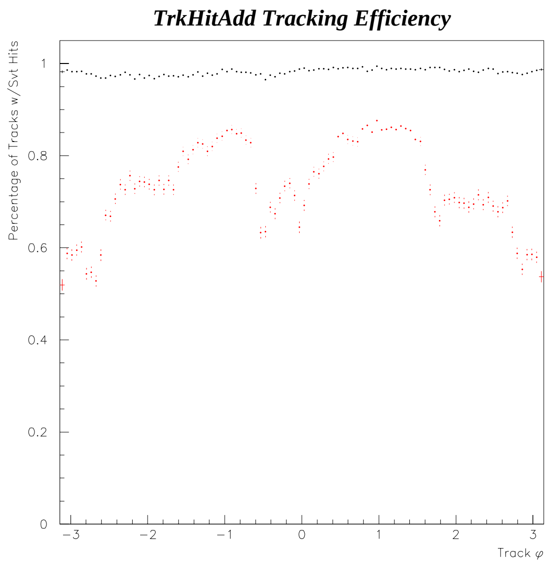

Figure 2: Efficiency for adding SVT information by Track Azimuth, black points are `TrkHitAdd`, red points are stand alone SVT tracking

In commissioning `TrkHitAdd` it was found that the penalty function needed to be carefully tuned to reduce the quantity of fake hits associated before the production vertices in tracks coming from Kaon decays and photon conversions. This fake rate is now down to a few %.

`TrkHitAdd` has also been studied as a way to extend low-momentum standalone SVT tracks back into the DCH. This algorithm is nearly ready to be deployed. `TrkHitAdd` has also been used to add SVT hits to both ends of low transverse momentum tracks which loop, and A strong piece of evidence for the usefulness and reuseablity of this implementation of Kalman filtering is the fact that none of this extensive set of changes and upgrades required any fundamental modification of the simple interfaces described above.

8 Fitting Two-Track Events for Detector Alignment

We have implemented an extension of the BaBar track fit that allows events of the type $e^+e^- \rightarrow e^+e^-$ or $e^+e^- \rightarrow \mu^+\mu^-$ to be fit as a single (piecewise) track. The fit utilizes the known momenta of the incoming e^+e^- to relate the track parameters between the two tracks of the resultant e^+e^- (or $\mu^+\mu^-$) pair. The result of this fit is a single representation of the two-track event with optimal parameters and their covariance at any position along the track[4]. This two-track fit also takes full advantage of the basic BaBar Kalman track fit's ability to account for the effects of material interactions and magnetic field distortions.

The two-track events are used in BaBar for both the internal and relative the alignment of the detector's tracking elements. The Kalman filter track fitting algorithm is ideal for use in detector alignment for several reasons. First, an optimal set of parameters is provided at all points along the track, not just at the origin. This provides the most accurate estimate of the track position at an active detector element. Second, the effects of a hit in a detector element which one wants to align can easily be removed from the fit, producing a fit result which is unbiased by the detector element in question. How this is done follows from equation 6. One takes the weight-space average of the (cached) results from the outward processing of the site before and the inward processing of the site after the site being removed from the fit. Fitting the two tracks simultaneously adds the advantages of providing greater accuracy on the track's trajectory and producing a strong correlation between two tracks in opposite hemispheres of the tracking volume. This correlation helps to constrain certain systematic distortions of the detector during alignment, such as elliptical azimuthal compression.

8.1 Algorithm Description and Implementation

The two-track fits are represented by the class `KalPairRep` which inherits from `KalRep`. The constructor of this class takes as its input two tracks which have already been fit using the BaBar Kalman filter fitter, plus a vector and matrix which contain the known e^+e^- total 3-momenta and covariance, respectively. The input track corresponding to the negatively charged track (by convention) is inverted and prepended to the positively charged track, creating a new "pair" track. The inversion process reverses the order of the sites within the track creating a track that appears to travel in the opposite direction and with the opposite charge. Each site is aware of if it has been inverted so effects such as energy loss and magnetic field inhomogeneities can be corrected for with the proper sign. The pair track then contains all of the sites from both the positive and negative tracks.

A concrete `KalSite` subclass `KalPairSite` is used to describe the effect of the known beam momenta on the track fit. This site is inserted in the `KalPairRep` at a flight length corresponding to the point of closest approach of the parent positive and negative tracks. The flight length and its error are calculated using the positive and negative tracks' parameters and covariance, with the assumption that the tracks' curvatures are small. The `KalPairSite` stores this flight length and its error[5], as well as the beam momenta and covariance.

Once the `KalPairSite` has been inserted in to the track, the new pair track can be fit by simply invoking the 'fit' function inherited from `KalRep`.

[4]This should be contrasted with the case where the two resultant tracks are individually fit, followed by a vertex and momentum constrained fit performed on the two tracks. In this case, one may obtain optimal parameters at the vertex, but not at every point along the track.

[5]The KalPairSite actually requires two flight lengths, l_+ and l_-, corresponding to l along the positive and negative track, respectively, at the point of closest approach between the two tracks. The two flight lengths are used during the inward (l_+) and outward (l_-) processing.

`KalPairSite` implements its ‘process’ function in parameter space. The parameters, P, and flight length of the `KalPairSite` are used to calculate the position vector, $\mathbf{x}$, and 3-momentum, $\mathbf{p}$. The new position and momentum of the track are then given by

$$\begin{aligned} \mathbf{x}' &= \mathbf{x} \\ \mathbf{p}' &= \mathbf{p} + \begin{cases} +\mathbf{p}_{Beam} & \text{outwards} \\ -\mathbf{p}_{Beam} & \text{inwards} \end{cases}, \end{aligned} \tag{10}$$

where $\mathbf{p}_{Beam}$ is the total incoming e^+e^- momentum. The processed parameter vector, P', is then determined from these new position and momentum vectors.

The covariance transformation involves three separate contributions. The first comes from the Jacobian determined by the parameter transformation $P \rightarrow P'$. If we define the matrix $J(q,r)$ to have elements given by

$$J(q,r)_{i,j} \equiv \frac{\partial q_i}{\partial r_j}, \tag{11}$$

then

$$C \rightarrow C' = J(P',P)CJ(P',P)^T \tag{12}$$

The second contribution to C comes from the uncertainty on the beam momentum, $\mathbf{p}_{Beam}$, expressed by the covariance C_{Beam}:

$$C \rightarrow C'' = C' + J(P',\mathbf{p}_{Beam})C_{Beam}J(P',\mathbf{p}_{Beam})^T \tag{13}$$

The third contribution to C comes from the uncertainty on the location of the pair site, expressed by the error on the flight length, σ_l, determined during the point of closest approach calculation described previously:

$$C \rightarrow C''' = C'' + J(P',l)\sigma_l^2 J(P',l)^T \tag{14}$$

All derivatives are calculated analytically. Note that in the limit $\mathbf{p}_{Beam} \rightarrow 0$, the operations defined by equations 10, 12 and 14 are null operations. This would be the case if PEPII were a symmetric e^+e^- collider.

8.2 Applications in BaBar

Two-track Kalman fits are currently being used in BaBar for two different applications, both having to do with the alignment of the BaBar tracking detectors.

The first application that makes use of the two-track fits is the internal alignment of the 340 silicon wafers in the inner tracker. The BaBar silicon detector consists of 5 layers of double sided silicon giving each track typically 10 measurements. Ideally, one would like to perform the alignment of the silicon detector without relying upon measurements from any other tracking devices. Because the outer radius of the silicon detector is only $\sim 14\,\mathrm{cm}$, the resolution on the curvature [6] of tracks from $e^+e^- \rightarrow \mu^+\mu^-$ events is only $\sim 17\%$ if only the silicon detector is used. However, by using the two-track fit, this resolution improves by more than 2 orders of magnitude to $\sim 0.15\%$. The curvature resolution from the two-track fit is compared to the resolution from single track fits in Figure 3 for $e^+e^- \rightarrow \mu^+\mu^-$ Monte Carlo. This improvement comes about mostly because the silicon detector provides very accurate measurements of the track position and

[6] See Appendix A for a definition of the track parameters used in BaBar

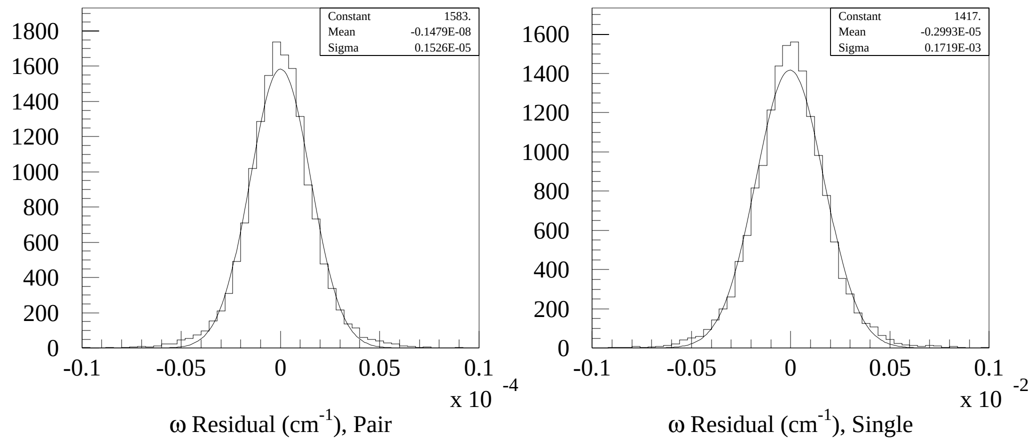

Figure 3: Curvature resolution from the two-track fit. Note that the x-axis scale is different by 2 orders of magnitude between the two figures.

angles, and the off-diagonal elements of $J(P', P)$ in equation 12 couple this information into the curvature. This would not be true if BaBar operated in a symmetric collider, i.e. $\mathbf{p}_{Beam} = 0$. Therefore, the two-track fit allows one to use information solely from the silicon detector and achieve tracking resolutions that are actually better than what one would obtain from single track fits using both the silicon and the drift chamber. Furthermore, the two-track fit generates a strong tie between elements of the silicon detector in opposite hemispheres. This constrains possible distortions that would otherwise be difficult measure. Monte Carlo studies show that one can obtain about a factor of 5 improvement in the alignment accuracy using the two-track fit with the same statistical sample of data.

The second application of the two-track fit in BaBar is for doing the relative alignment between the silicon detector and the drift chamber. In this case, the pair is fit twice: once using only the hits from the silicon detector and a second time only using the hits from the drift chamber. The values of the track parameters from the two fits are compared and a fit for the six parameters (three translations and three rotations) describing the alignment of the silicon to the drift chamber is performed. Again, Monte Carlo studies have shown that the alignment accuracy using this method is better by about a factor of 5 compared to single track fits with the same statistical sample.

9 Other Features of the BaBar Kalman Fit

The BaBar implementation of the Kalman fit provides some special features useful for specific purposes. These exploit the simple, symmetric structure of the fit to provide precise values with an efficient implementation.

The BaBar fit allows parameters to be constrained through a special `KalRep` constructor (see section 4). This takes as additional input a description of which columns of the reference parameter vector are to be constrained. The reference covariance matrix terms for those columns are used as-is (ie not increased by the usual factor of 10^6) when seeding the fit, effectively constraining the reference values. This option is used for some alignment procedures in BaBar, fitting the SVT hits constrained to the curvature measured in the DCH.

Work is also in progress on an algorithm to detect decays in flight using the Kalman fit, using a dedicated `KalSite` subclass to represent the physics model of the decay. A `KalSite` subclass representing a misalignment between the tracking chambers is also being developed. An algorithm to detect and remove hit outlyers based on the Kalman fit information is also under development.

10 Conclusions

The BaBar implementation of the Kalman filter track fit is a central part of BaBar reconstruction, and has performed well during our first data taking period. The flexibility of the Kalman formalism and the Object Oriented design of the implementation have allowed its extension to pattern recognition, alignment, and other tracking-related functions.

A The BaBar Track Trajectory Parameterization

BaBar tracks are nominally described as a helix about the z (magnetic field) axis, defined by the following parameters:

$$P \equiv \left(d_0, \quad \varphi_0, \quad \omega, \quad z_0, \quad \tan\lambda \right) \tag{15}$$

The parameters can be interpreted in terms of the track's point of closest approach to the origin in the $x-y$ plane: d_0 is the distance of closest approach, signed by the angular momentum at that point, φ_0 is the angle in the $x-y$ plane at closest approach, and z_0 is the distance from the closest approach to the origin. in the z projection. the parameter ω may be interpreted as the $x-y$ plane curvature of the track, and $\tan\lambda$ as the tangent of the track dip angle in the $\rho-z$ projection. The position of the particle as a function of the $x-y$ plane projection of the flight length from the point of closest approach l is given by:

$$\vec{F}(P:l) = \begin{cases} \left(\frac{\sin(\varphi_0+\omega l)}{\omega} - (1/\omega + d_0)\sin\varphi_0\right)\hat{x} \\ \left(-\frac{\cos(\varphi_0+\omega l)}{\omega} + (1/\omega + d_0)\cos\varphi_0\right)\hat{y} \\ (z_0 + l\tan\lambda)\,\hat{z} \end{cases} \tag{16}$$

The momentum $\vec{p}(l)$ of the track may be expressed from this, in terms of the charge of the particle q and the z component of the magnetic field B_z:

$$\vec{p}(l) = \frac{qcB_z}{\omega}\begin{cases} \cos(\varphi_0+\omega l)\hat{x} \\ \sin(\varphi_0+\omega l)\hat{y} \\ \tan\lambda\hat{z} \end{cases} \tag{17}$$

B The BaBar Track Parameter Derivatives

The parameter derivatives used in the BaBar fit can be derived by considering how a small change in a physical parameter of the track (such as its direction or energy) affects the track parameters. Changes are considered for two directions, the deflection angle normal to the track in the $\rho-z$ plane (Θ), and the deflection angle normal to the track in the $x-y$ plane (Φ). We also consider a change in the momentum fraction $\Psi \equiv \Delta p/|p| = -\Delta\omega/\omega$.

The derivatives are calculated by starting with equations 16 and 17, and considering a small change in one of Θ, Φ, or Ψ, holding the others fixed and requiring that the track remain continuous in space. This results in six equations for six unknowns (the five new track parameters plus the new flight length) for Θ, Φ, and Ψ, which can be solved analytically. The derivative of those equations (as a function of transverse flight length l) are presented below. These equations have been tested through comparison with numerically estimated derivatives.

$$\frac{\delta P}{\delta\Theta} = \left(\tfrac{\tan\lambda(1-\cos(\omega l))}{\omega},\ \ -\tfrac{\tan\lambda\sin(\omega l)}{(1+\omega d_0)},\ \ \omega\tan\lambda,\ \ -\tfrac{\tan^2\lambda\sin(\omega l)}{\omega(1+\omega d_0)} - l,\ \ \tfrac{1}{\cos^2\lambda} \right) \quad (18)$$

$$\frac{\delta P}{\delta\Phi} = \left(-\tfrac{\sin(\omega l)}{\omega\cos\lambda},\ \ \tfrac{\cos(\omega l)}{\cos\lambda(1+\omega d_0)},\ \ 0,\ \ -\tfrac{\tan\lambda}{\omega\cos\lambda}(1 - \tfrac{\cos(\omega l)}{(1+\omega d_0)}),\ \ 0 \right) \quad (19)$$

$$\frac{\delta P}{\delta\Psi} = \left(\tfrac{(1-\cos\omega l)}{\omega},\ \ \tfrac{\sin(\omega l)}{(1+\omega d_0)},\ \ -\omega,\ \ -\tan\lambda\left(l - \tfrac{\sin(\omega l)}{\omega(1+\omega d_0)}\right),\ \ 0 \right) \quad (20)$$

References

1 D. Brown, talk presented at CHEP97, http://www.ifh.de/CHEP97/abstract/a341.htm

2 ”BaBar Technical Design Report”, SLAC-REP-950457 (1995)

3 ”An Asymmetric B factory based on PEP: Conceptual design report” SLAC-0372 (1991)

4 P. Billoir, NIM A225 (1984) 352
J. Boudreau, ’The SLT Class Library’ http://cactus.phyast.pitt.edu/ joe/slt.html

5 R. Fruewirth, NIM A262 (1987) 444

6 The authors wish to acknowledge Ian J. Scott and Natalia V. Kuznetsova as full collaborators in the design and implementation of the central `TrkHitAdd` software, as well as in the implementation and testing of the DCH to SVT application of `TrkHitAdd`. We would also like to thank Michael H. Kelsey for the implementation of the SVT to DCH application of `TrkHitAdd`, as well as for the numerous improvements made to the design and interfaces of the core `TrkHitAdd` software during that process.

7 S. Schaffner, talk presented at CHEP97 http://www.ifh.de/CHEP97/abstract/a369.htm

8 C. Caso etal,”Review of Particle Physics” Euro. Phys. Jour. C3 (1998) 1
R. M. Sternheimer, M. J. Berger and S. M. Seltzer, “Density Effect For The Ionization Loss Of Charged Particles In Various Substances,” Atom. Data Nucl. Data Tabl. **30**, 261 (1984).

9 ”The GEANT4 Program Library”, http://wwwinfo.cern.ch/asd/geant4/reports/reports.html