KinKal: A Kinematic Kalman filter Track Fitting Package for the Mu2e Experiment

David Brown[1,*]

[1]Lawrence Berkeley National Lab

Abstract. Many current physics experiments require precision reconstruction of low-energy particles. One example is the Mu2e experiment, which requires reconstructing an isolated 105 MeV electron with better than 500 KeV/c momentum resolution. Mu2e uses a low-mass straw tube tracker, and a CsI crystal calorimeter, to reconstruct tracks. In this paper, we present the design and performance of a track reconstruction algorithm optimized for Mu2e's unusual requirements. The algorithm is based on the KinKal [1] kinematic Kalman filter track fit package. KinKal supports multiple track parameterizations, including one optimized for looping tracks, such as Mu2e signal tracks, and others optimized for straight or slightly-curved tracks, such as the high-momentum (>1 GeV/c) cosmic ray muons used to calibrate and align the Mu2e detectors. All KinKal track parameterizations include the track origin time, to correctly model correlations arising from measurements that couple time and space, such as the straw drift time or the calorimeter cluster time. KinKal employs magnetic field inhomogeneity and material effect correction algorithms with 10^{-4} fractional precision. The Mu2e fit uses Artificial Neural Net functions to discriminate background hits from signal hits, and to resolve the straw tube hit left-right ambiguity, while iterating the extended Kalman filter. The efficiency, accuracy, and precision of the Mu2e track reconstruction, as tested on detailed simulations of Mu2e data, are presented.

1 Introduction

Kinematic particle fitting combines the extraction of geometric and kinematic properties of a particle from measurements. It was developed to provide simultaneous particle identification and momentum reconstruction in bubble chamber experiments over 50 years ago [2], and is required for precision track reconstruction when a particle's energy is comparable to its mass. In a kinematic fit energy and time are intrinsic properties of the fit. A kinematic fit provides a natural framework for coherently combining time, position, and coupled time-and-position (such as drift cell) sensor constraints.

The Kalman filter algorithm has been used to reconstruct tracks in particle physics experiments for over 30 years [3]. Most recent developments have been focused on the need for rapid reconstruction of large volumes of multi-track events recorded by high-intensity collider experiments. Modern Kalman filter track fits apply corrections for material effects (energy loss and scattering) and inhomogeneous magnetic fields through transport equations. They also provide interfaces for pattern recognition and detector calibration and alignment.

*e-mail: dave_brown@lbl.gov

In this paper we present KinKal [1], a kinematic Kalman filter track fitting package designed for the precise reconstruction of low-momentum tracks. KinKal was developed for the Mu2e experiment [4], which will search for the neutrino-less conversion of negative muons into electrons and positrons. Mu2e has several unique track fitting requirements:

- Signal and physics background events consist of a single particle with momentum near the muon mass (105 MeV/c), in the presence of large amounts of pileup detector backgrounds
- There is no a-priori constraint on the origin time of signal or physics background particles, as they are produced from the random decay of a muon
- The origin position of signal and physics background particles is only weakly constrained to a region 7.5 cm in diameter by 50 cm long
- The track fit must function in regions of large magnetic gradient, including fitting particles which reverse direction
- Achieving a sensitivity of $\sim 10^{-16}$ places strong limits on the non-Gaussian tails of the reconstructed momentum resolution
- Signal particles execute multiple loops in the tracker, requiring a track fit that accommodates multiple loops for optimal fits
- In-situ calibration and alignment of the tracker requires reconstructed high-energy cosmic ray muon tracks, best described using a traditional HEP impact-parameter based parameterization
- Traditional impact-parameter based helix parameterizations become degenerate describing low-energy cosmic ray particles co-axial with the solenoid, which are required to constrain the 'twist' miss-alignment weak mode of the tracker
- Commissioning data for Mu2e will be taken without any magnetic field, requiring reconstructing straight tracks for calibration, alignment, and testing
- Optimal calibration and alignment requires combining the different track types described above, requiring a tracking algorithm that can simultaneously describe tracks with different parameterizations

The KinKal (Kinematic Kalman) track fitting package was developed to satisfy these requirements. Though designed primarily to serve the needs of Mu2e, it is a standalone tracking package that could be used in other applications requiring precision low-momentum tracking.

2 Overall Software Design

KinKal is divided into several libraries, each describing a particular aspect of the fit; particle trajectories, geometry, detector descriptions, fit configuration, and the fit engine itself. A test library defines a complete set of unit tests of all the low-level classes, as well as a high-level test of the fit using a toy Monte Carlo and an example detector description. Detailed sensor and material classes must be added to these base libraries to create a functional fit for a specific experiment. These core KinKal libraries are described in detail below.

3 Kinematic Trajectory

The core class in KinKal represents the kinematic trajectory of a particle in a fixed magnetic field. A kinematic trajectory class must implement an interface implicitly defined by the KinKal fit. They must describe the position and momentum of a particle as a function of

absolute time, over a limited time range, defined by a set of 6 parameters. The kinematic trajectory class must provide a complete set of derivatives of the position and momentum relative to the parameters, the inverse of those, and the derivatives of the parameters with respect to a change in the reference magnetic field. Additional functions giving the particle energy, velocity, and related values, as a function of time are also required. The kinematic trajectory class payload consists of the parameters, described below (section 4), the reference magnetic field, and the particle mass. These last are fixed (non-parametric) members of the class. Kinematic trajectory class objects can be losslessly converted to and from a particle description as a position and momentum at a fixed time, with the full 6X6 covariance matrix of those components.

A kinematic trajectory's geometric parameters are not proscribed by KinKal. The parameterization must include a time t_0, representing the physical time when the particle passes a well-defined but arbitrary reference location. For example, t_0 may be defined as when the particle passes through a particular plane, or comes closest to a particular axis.

KinKal provides three fully implemented kinematic trajectory classes. The *LoopHelix* class is optimized for low-momentum particles which execute multiple loops in a magnetic field, describing the loop center, transverse radius, and longitudinal wavelength. The *CentralHelix* class uses a traditional HEP high-momentum helix parameterization based on impact parameter, directions, and inverse curvature. The *KinematicLine* class describes a straight trajectory (no magnetic field), by its impact parameters and directions, plus a scalar momentum value. The KinKal kinematic trajectory classes are illustrated in figure 1. A particular fit uses exactly one kinematic trajectory class.

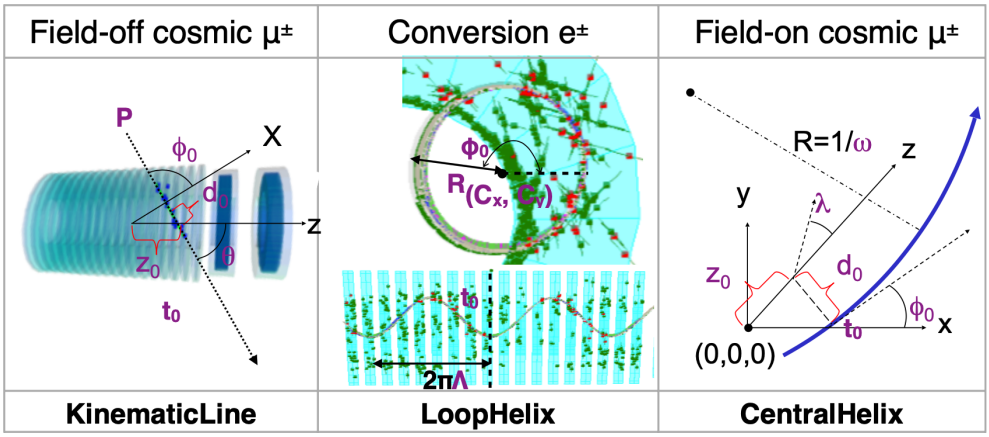

Figure 1. KinKal Kinematic Trajectory Parameterizations

4 Parameters and Weights

The parameteric content of a kinematic trajectory is held as a Parameter class, which combines a 6-dimensional ROOT::SVector P and a 6X6 symmetric ROOT::SMatrix C for the covariance. These classes provide very efficient implementations of 6-dimensional linear algebra functions such as similarity and inversion. A Parameter object is generic and algebraic, its physical interpretation given only through the kinematic trajectory which owns it.

Parameter objects may be losslessly transformed into Weight objects, with the identical payload as a Parameter, according to the definition:

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

where β and γ are the SVector and symmetric SMatrix of the Weight, respectively.

5 Geometry

The KinKal Geometry library provides classes describing simple geometric objects that are used to implement the detector representation classes described below. It also includes classes that compute geometric relations between these and the kinematic trajectory classes.

The Geometry library provides classes representing points and line segments in space, and classes for calculating the position and time of closest approach (TPOCA) of a kinematic trajectory to those. The TPOCA class provides full analytic derivatives of the change in TPOCA against the kinematic trajectory parameters.

The Geometry library also provides classes representing simple bounded surfaces, such as segments of planes, cylinders, and cones. It also provides an Intersection class that can find the time and position of a geometric intersection of a kinematic trajectory with a surface segment.

6 Detector Representation

The KinKal fit is driven off input objects representing detector measurements (hits), which constrain the fit, and material interactions, which degrade and randomize the fit. Hit and material interaction classes are experiment-dependent and must be implemented by experiments according to their physical apparatus. To be used in the KinKal fit the hit class must provide a function computing its constraint (parameter difference vector and weight matrix) with respect to a reference kinematic trajectory. It must also provide a function to update its internal state based on a revised estimate of the kinematic trajectory.

Similarly the material interaction class must provide a function to compute its effect on a reference kinematic trajectory (parameter vector change and covariance matrix increase), and an update function for recomputing its internal state.

A generic hit class based on *residuals* is provided, where a residual is defined as a one-dimensional constraint, with corresponding variance and dependence on kinematic trajectory parameters. Similarly, a generic material class based on a material in the form of a thin surface section is provided.

Several fully implemented example detector classes are provided in the KinKal Examples library. For instance, a residual hit subclass based on a TPOCA constraint to a line segment representing the wire in a drift cell is provided. A purely time-based residual subclass based on measurement of a time in a detector with linear signal propagation (such as a scintillator bar or crystal), and a measurement class making a direct Gaussian constraint on a subset of parameters are also provided.

7 Updaters

Measurements and material interactions may need to be updated as the fit progresses, as the estimate of their internal state or relevance to the fit may change. Updating is performed by dedicated *Updater* classes, associated with a corresponding detector class. Thus a drift cell hit subclass has an associated updater for fixing the left-right ambiguity. Any number

of Updaters may be associated with a given detector class, allowing factorization of different effects, such as outlier removal, drift calibration, etc. Specific Updater class instances are associated with specific *meta-iterations* of the fit, as defined in the configuration section below (section 10).

8 Magnetic Field

KinKal requires a magnetic field map class which can provide the field value vector and gradient tensor at a given position. This may be implemented as a simple wrapper around (say) a Geant4 magnetic field map. Fits of KinematicLine objects use a null field.

The helical trajectory parameterizations in KinKal implicitly account for particle motion in a constant magnetic field. To account for spatial variations in the magentic field KinKal divides the particle path into many contiguous segments or *domains*, each assuming a constant local magnetic field. The domains are defined relative to a seed trajectory, which defines initial parameters and particle path.

Domains are defined to keep the residual distortion of the momentum derived from the fit within a tolerance specified by the fit configuration. Starting from one temporal end of the seed, the domain are set by calculating the maximum the particle can proceed along the seed trajectory, keeping the fractional difference in the momentum calculated using the initial (fixed) field from the momentum calculated using the actual field within tolerance. The end of that domain and the start of the next is set to that time, and the process continues till the calculation reaches the end of the range of the seed.

A smaller tolerance value translates into finer segmentation (more domains). Similarly, there are more, smaller domains in regions of rapid field change compared to regions of more constant field. Having an adjustable tolerance on the magnetic field correction allows the end user to make application-specific optimization of computing performance vs fit accuracy.

9 Kalman Filter Implementation

The KinKal Track class implements the Kalman filter fit and records its output. Track is templated on a kinematic trajectory, and so can be used to fit using any of them. All kinematic and geometric functions in Track are implemented as call-downs to kinematic trajectory functions, or detector objects through their adapters.

The Track constructor takes the hits and material interaction objects, wrapping the detector-specific classes with generic adapters that interface directly with the fit implementation. It also takes the magnetic field map, and the seed estimate of the kinematic trajectory, which is used to compute the magnetic domains defined above. The constructor also takes a configuration object, which specifies the annealing schedule and the configuration of the detector class updaters. The fit is performed on construction. Tracks may be extended after construction to include additional hits, additional materials, or changes in the configuration.

The effect of magnetic field domains on the fit is implemented by the Domain class, which records the domain boundaries and reference magnetic field, defined at the domain centers. Domain objects are added to the fit on construction, based on the configuration parameters defined below in section (10). At domain boundaries, the fit parameters are transported so as to represent the same physical position and momentum, relative to the new domains magnetic field. The transport implementation uses the analytic derivatives of the parameters with respect to magnetic field change provided by the kinematic trajectory classes.

The fit is implemented as an extended Kalman filter fit, performed as a nested set of iterative loops. The inner loop performs Kalman filtering, repeated to algebraic convergence (or

divergence) in both forwards and backwards time direction. The effect of material transport is reversed in reverse-time filtering; all other transport is symmetric. The internal state of all detector objects are held constant during the inner loop, but the derivatives are re-computed each iteration using the updated fit estimate. The outer (meta-iteration) loop updates the internal state of the detector objects and magnetic domains according to the configuration settings and updaters associated with that particular meta-iteration, and applies the annealing factor to the hit errors, before repeating the inner loop.

The filtering is performed symmetrically in both forwards and reverse time direction, following the BaBar [6] Kalman filter fit implementation. The fit is initialized using the parameters at the appropriate end of the previous fit iteration, de-weighted by a configurable parameter. Hits, material interactions and domain boundary transits are then processed in time sequential order. Hits are processed in weight space by adding their constraint to the current fit Weight. Material interactions and domain boundary transits are processed in parameter space. The fit uses lazy evaluation accessors to minimize the transitions between weight and parameter space.

After filtering has converged, the Track class constructs an estimate of the complete particle trajectory as a collection of kinematic trajectories, whose individual time ranges contiguously covers the full fit time range of the relevant data used to drive the fit. This piece-wise kinematic trajectory breaks occur wherever the parameters change physically due to material interactions, or where they change due to crossing a domain boundary. This fit is used as reference for the next meta-iteration.

Fit status information including the convergence status and the fit chi-squared consistency, is stored for each meta-iteration in the Track, providing the final status and the full convergence history. The piece-wise kinematic trajectory computed at the end of the last meta-iteration is stored in the Track object, providing downstream users with a coherent estimate of the particle's spatial, temporal, and kinematic information, with covariance, at any physical time.

10 Configuration

The fit behavior is defined by a Configuration object, which defines the annealing schedule, the meta-iteration schedule, the convergence and divergence criteria, and the fractional momentum accuracy tolerance. The main Configuration object holds a vector of MetaConfig objects, each configuring a single meta-iteration. MetaConfig contains the annealing temperature as well as the Updater objects specific to that meta-iteration.

Updater objects are attached to specific MetaConfig instances using the std::any paradigm. The MetaConfig object for a specific meta-iteration is passed to every object in the Track during the update phase of that meta-iteration. Detector objects in the fit search for relevant updaters inside the MetaConfig object in their update method, and if found, use those algorithms and parameters to update their internal state. For instance, the machine learning (ML) algorithm used to filter background straw hits in the Mu2e KinKal fit described below is implemented as an updater associated with the Mu2e::StrawHit detector class (see section 11).

11 Mu2e KinKal Implementation

Mu2e implements hit and material detector classes to represent the straws in our straw tracker. The straw hit provides a spatial residual based on the straws position and (optionally) drift information, and a temporal residual based on the independent time-over-threshold measurement.

The Mu2e straw material interaction class describes the effect of a particle traversing the drift gas and straw wall. Straw material instances and straw hit instances in the same straw are linked, such that the material effect is ignored if the straw hit is inactive in the fit. Straw material instances for inefficient or dead straws are added between the initial , by using the trajectory to find straws intersected without a hit. The amount of gas and aluminized mylar traversed by a particle is estimated using the distance of closest approach between the straw and the current fit estimate.

Mu2e also implements a hit class based on timing information obtained from clusters measured in the CsI calorimeter. This hit is implemented as a time residual hit using a line segment parallel to the crystal axis, positioned at the cluster centroid in the transverse direction, similar to how calorimetry information was used in BaBar [5]. This technique intrinsically corrects for the propagation time of the light to the SiPM, and provides an optimal constraint on the t_0 parameter in the fit.

Mu2e uses several Machine learning algorithms to define the internal state of the straw hits. One is trained to separate hits from background particles from those belonging to the track, deactivating the background hits in the next fit iteration. Another is trained to determine when the left-right ambiguity measured using the current fit estimate is accurate. Another is trained to determine when the ionization statistics in a given straw are likely to give an accurate estimate of the drift radius given the drift time. These are used together to decide if the drift information is accurate enough to constrain the fit to the left-right ambiguity-signed drift radius (with small errors), or to the wire position (with larger errors). The background hit rejection and drift quality algorithms are implemented as separate updaters, which can be combined as needed in a given meta-iteration.

Mu2e uses a least-squares fit to 3-D hit straw positions to seed the KinKal fit. The initial fit uses only the hits associated with that simple fit. After convergence, that fit trajectory is used to search for additional hits and straw materials, and additional calorimeter clusters that might have been missed by the least-squares fit. The fit is then extended to add those.

Mu2e has implemented a text-based interface to the Configuration object, allowing the iteration parameters and updater algorithms to be specified in human-readable and editable text.

Mu2e plans to use a KinKal fit based on the LoopHelix kinematic trajectory to search for muon to electron conversion, with a fractional momentum tolerance of 10^{-4} and 12 meta-iteration steps. This fit has been tested using a realistic Geant4-based simulation [7], which showed that the fit meets the Mu2e scientific requirements. Mu2e has successfully used the KinematicLine KinKal fit on cosmic rays observed in a sub-assembly of the track.

Mu2e plans to use a LoopHelix KinKal fit in its realtime event selection (trigger), which will run on a dedicated server farm. Resource limitations require that fit be very efficient. The trigger fit is configured to use a modest (10^{-2}) fractional momentum tolerance, which we find saves an order of magnitude in processing time compared to the analysis quality fit. The trigger fit does not add hits or materials.

12 Dependencies

KinKal uses root [8] SMatrix package ([9]) to implement its matrix algebra. It uses the root GenVector package [10] to describe spatial coordinates and vectors. The ANNs are trained using Keras [11] and TensorFlow [12]. The ANN models are converted to C++ code using the root SOFIE package [13].

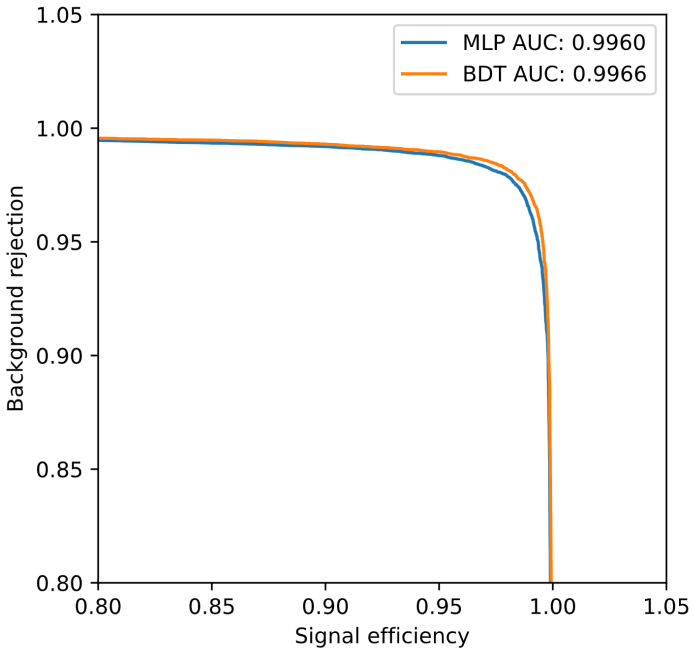

Figure 2. ROC curve from the TensorFlow training of the background hit filtering ANN used in the Mu2e KinKal track fit straw hit updater

References

[1] KinKal, a Kinematic Kalman Filter Track Fit, github.com/KFTrack/KinKal

[2] **Bubble Chamber Film Analysis**, I. R. Kenyon, Contemporary Physics, 13:1, 75-104 (1972)

[3] R. Fruewirth, NIM A262 (1987) 444

[4] Mu2e Technical Design Report, L. Bartoszek et al. [Mu2e Collaboration]. FERMILAB-TM-2594, FERMILAB-DESIGN-2014-01

[5] Extracting longitudinal shower development information from crystal calorimetry plus tracking, D. N. Brown et. al., Nucl. Instrum. Meth. A **592**, 254 (2008)

[6] The BaBar Track Fitting Algorithm, D. N. Brown et. al., presented at CHEP 2000

[7] Geant4—a simulation toolkit, S. Agostinelli et. al., Nucl. Instrum. Meth. A **506** 3 (2003)

[8] ROOT - An Object Oriented Data Analysis Framework, Rene Brun and Fons Rademakers, Nucl. Inst. Meth. in Phys. Res. A **389** (1997) 81-86

[9] Root SMatrix Package, https://root.cern.ch/doc/v608/SMatrixPage.html

[10] Root GenVector Package, https://root.cern/doc/v612/Vector.html

[11] Keras, Chollet, François and others, 2015, https://keras.io

[12] TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems, Martín Abadi et. al., 2015, tensorflow.org

[13] New developments of TMVA/SOFIE, L. Moneta et. al, presented at CHEP 2023