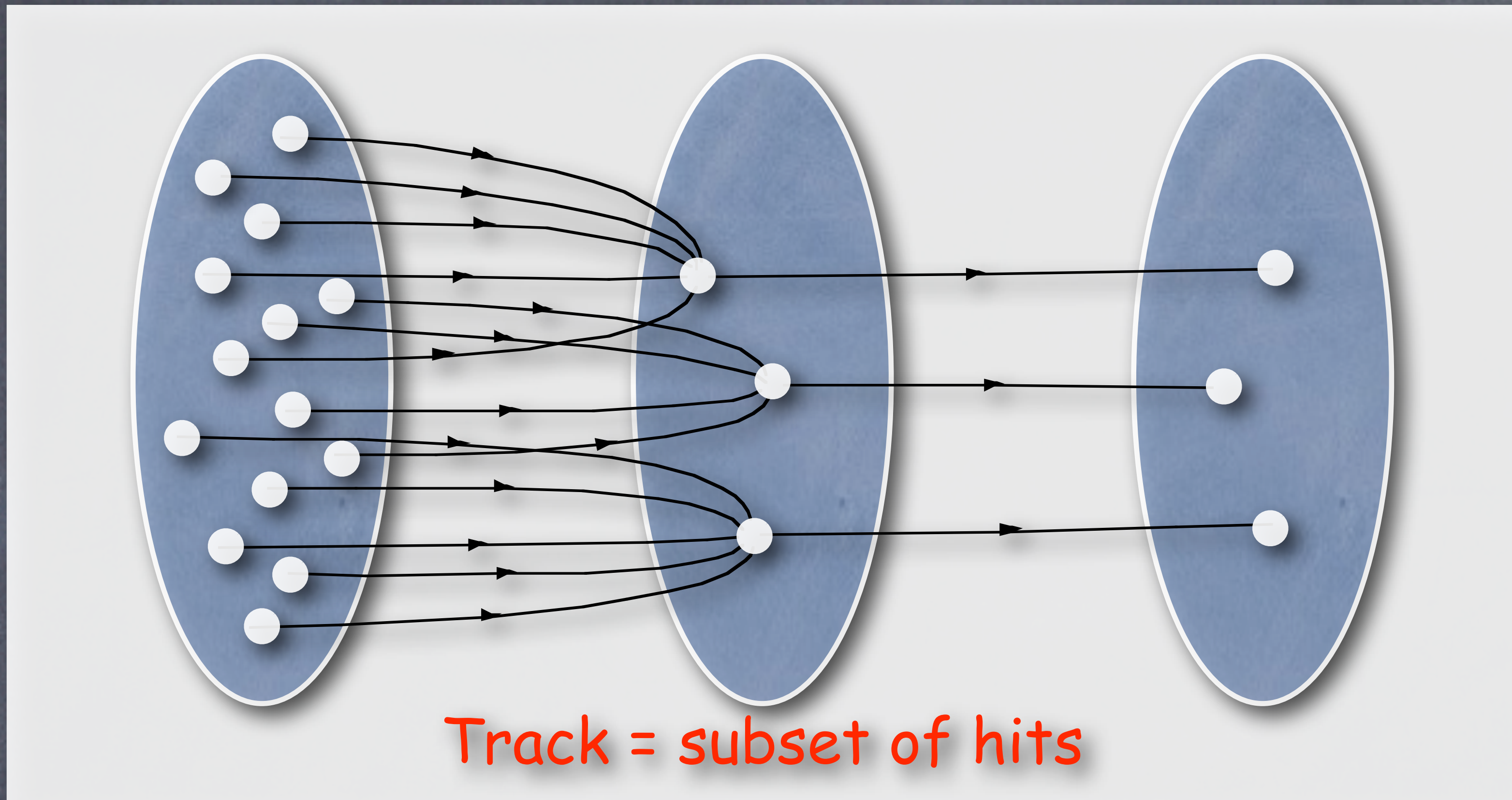
KalTest

-- Extended Kalman Filter --

Kiesuke Fujii, KEK

April 16, 2009

Tracking = Track Fitting x Track Finding



Set of Hits → Set of Tracks → Set of Track Parm. Vectors

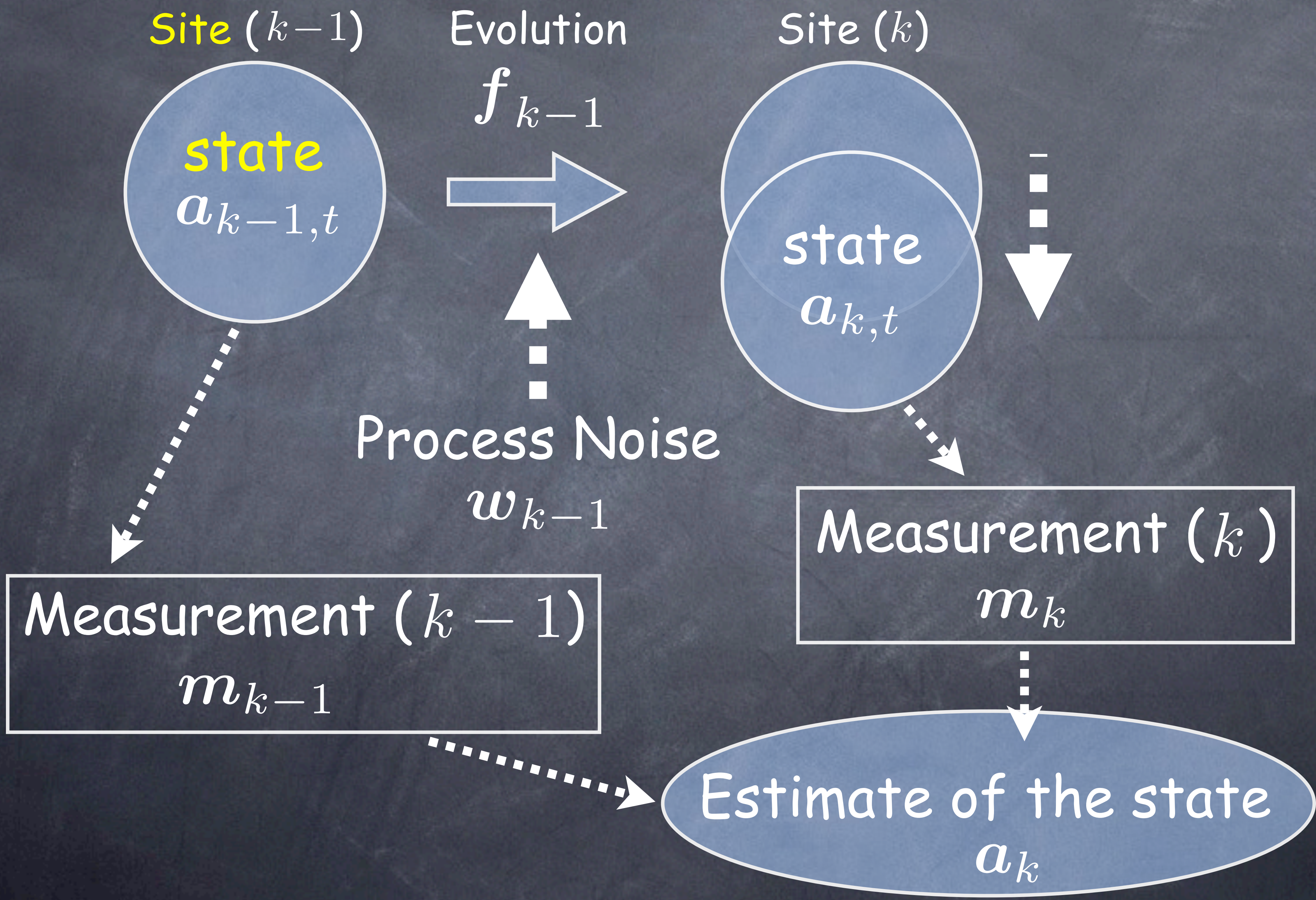
Track Finding — Track Fitting

Today's menu

Kalman Filter Formulation	Its C++ Implementation	Application to ILC Trackers

Statement of the Problem

System



Site $(k-1)$
state
$\boldsymbol{a}_{k-1,t}$
Evolution
$\boldsymbol{f}_{k-1}$
Site (k)
state
$\boldsymbol{a}_{k,t}$
Process Noise
$\boldsymbol{w}_{k-1}$
Measurement (k)
$\boldsymbol{m}_k$
Measurement $(k-1)$
$\boldsymbol{m}_{k-1}$
Estimate of the state
$\boldsymbol{a}_k$

Example 1 : Ballistic Missile (Original Application)

$$\boldsymbol{a}_k = \begin{pmatrix} \boldsymbol{x} \\ \boldsymbol{p} \end{pmatrix}_k$$

Position and momentum at (k)

$$\boldsymbol{w}_{k-1}$$

Random turbulence between ($k-1$) and (k)

$$\boldsymbol{m}_k$$

position and velocity measured
with a radar at (k)

$$\boldsymbol{\epsilon}_k$$

Measurement error of radar

Track = as a Kalman System consisting of Sites (Hits)

state

$$\boldsymbol{a}_k = \begin{pmatrix} d_\rho \\ \phi_0 \\ \kappa \\ d_z \\ \tan\lambda \end{pmatrix}_k$$

$\boldsymbol{w}_{k-1}$

Helix parameter vector at (k)

Multiple scattering between ($k-1$) and (k)

site

$\boldsymbol{m}_k$

$\boldsymbol{\epsilon}_k$

Measured hit point at (k)

random detector noise

System Equation (Equation of Motion)

$$\boldsymbol{a}_{k,t} = \boldsymbol{f}_{k-1}(\boldsymbol{a}_{k-1,t}) + \boldsymbol{w}_{k-1}$$

process noise from $(k-1)$ to (k)

true state vector at $(k-1)$

true state vector at (k)

Assume that process noise is random and unbiased

$$\begin{cases} \langle \boldsymbol{w}_k \rangle & = & \boldsymbol{0} \\ \langle \boldsymbol{w}_k \boldsymbol{w}_k^T \rangle & \equiv & \boldsymbol{Q}_k \end{cases}$$

Measurement Equation

$$\boldsymbol{m}_k = \boldsymbol{h}_k(\boldsymbol{a}_{k,t}) + \boldsymbol{\epsilon}_k$$

measurement noise

true measurement vector
at Site (k)

measurement vector at Site (k)

Assume that measurement
noise is random and unbiased

$$\begin{cases} \langle \boldsymbol{\epsilon}_k \rangle & = & \boldsymbol{0} \\ \langle \boldsymbol{\epsilon}_k \boldsymbol{\epsilon}_k^T \rangle & \equiv & \boldsymbol{V}_k \equiv \boldsymbol{G}_k^{-1} \end{cases}$$

What We Need = Recurrence Formulae

Machineary to do:

(i) Prediction

$$\{\boldsymbol{m}_{k'}; k' \leq k\} \mapsto \boldsymbol{a}_{k''>k} \quad : \text{ future}$$

(ii) Filtering

$$\{\boldsymbol{m}_{k'}; k' \leq k\} \mapsto \boldsymbol{a}_{k''=k} \quad : \text{ present}$$

(iii) Smoothing

$$\{\boldsymbol{m}_{k'}; k' \leq k\} \mapsto \boldsymbol{a}_{k''<k} \quad : \text{ past}$$

Notation

$$
\left\{
\begin{array}{lll}
\boldsymbol{a}_k^i & : & \text{extimate of } \boldsymbol{a}_{k,t} \text{ using measurements up to } (i) \\
 & & \quad (\boldsymbol{a}_k^k \equiv \boldsymbol{a}_k \text{ for simplicity of notation}) \\
\boldsymbol{C}_k^i & : & \text{covariance matrix for } \boldsymbol{a}_k^i \\
 & & \quad \boldsymbol{C}_k^i \equiv \left\langle (\boldsymbol{a}_k^i - \boldsymbol{a}_{k,t})(\boldsymbol{a}_k^i - \boldsymbol{a}_{k,t})^T \right\rangle \\
\boldsymbol{r}_k^i & : & \text{residual} \\
 & & \quad \boldsymbol{r}_k^i \equiv \boldsymbol{m}_k - \boldsymbol{h}_k(\boldsymbol{a}_k^i) \\
\boldsymbol{R}_k^i & : & \text{covariance matrix for } \boldsymbol{r}_k^i \\
 & & \quad \boldsymbol{R}_k^i \equiv \left\langle \boldsymbol{r}_k^i \boldsymbol{r}_k^{iT} \right\rangle
\end{array}
\right.
$$

Prediction

$$\{\boldsymbol{m}_{k'}; k' \leq k\} \mapsto \boldsymbol{a}_{k''>k} \quad : \ \text{future}$$

- State Vector

$$\boldsymbol{a}_k^{k-1} = \boldsymbol{f}_{k-1}(\boldsymbol{a}_{k-1})$$

- Covariance Matrix

Extrapolation Error

Process Noise

$$\boldsymbol{C}_k^{k-1} = \underbrace{\boldsymbol{F}_{k-1}\boldsymbol{C}_{k-1}\boldsymbol{F}_{k-1}^T}_{\text{Extrapolation Error}} + \underbrace{\boldsymbol{Q}_{k-1}}_{\text{Process Noise}}$$

$$\boldsymbol{F}_{k-1} \equiv \left(\frac{\partial \boldsymbol{f}_{k-1}}{\partial \boldsymbol{a}_{k-1}}\right)$$

- Residual

$$\boldsymbol{r}_k^{k-1} \equiv \boldsymbol{m}_k - \boldsymbol{h}_k(\boldsymbol{a}_k^{k-1})$$

- Covariance Matrix

Extrapolation Error

$$\boldsymbol{R}_k^{k-1} = \boldsymbol{V}_k + \boxed{\boldsymbol{H}_k \boldsymbol{C}_k^{k-1} \boldsymbol{H}_k^T}$$

Measurement Noise

$$\boldsymbol{H}_k \equiv \left(\frac{\partial \boldsymbol{h}_k}{\partial \boldsymbol{a}_k^{k-1}} \right)$$

Filtering

$$\{\boldsymbol{m}_{k'}; k' \leq k\} \mapsto \boldsymbol{a}_{k''=k} \quad : \text{ present}$$

- State Vector

New Pull

$$\boldsymbol{a}_k = \boldsymbol{a}_k^{k-1} + \boldsymbol{K}_k \left(\boldsymbol{m}_k - \boldsymbol{h}_k(\boldsymbol{a}_k^{k-1})\right)$$

Kalman Gain Matrix

$$\begin{aligned} \boldsymbol{K}_k &\equiv \boldsymbol{C}_k^{k-1} \boldsymbol{H}_k^T \left(\boldsymbol{V}_k + \boldsymbol{H}_k \boldsymbol{C}_k^{k-1} \boldsymbol{H}_k^T \right)^{-1} \\ &= \boldsymbol{C}_k^{k-1} \boldsymbol{H}_k^T \left(\boldsymbol{R}_k^{k-1} \right)^{-1} \end{aligned}$$

already calculated in the prediction step

Covariance Matrix

$$\boldsymbol{C}_k = (\mathbf{1} - \boldsymbol{K}_k \boldsymbol{H}_k)\, \boldsymbol{C}_k^{k-1}$$

Equivalent but different Way: Weighted Mean Method

$$\boldsymbol{C}_k = \left[\left(\boldsymbol{C}_k^{k-1} \right)^{-1} + \boldsymbol{H}_k^T \boldsymbol{G}_k \boldsymbol{H}_k \right]^{-1}$$

$$\boldsymbol{K}_k = \boldsymbol{C}_k \boldsymbol{H}_k^T \boldsymbol{G}_k$$

Which to use depends on the dimensions of state vector and measurement vector

Improvement from New Measurement at (k)

- Residual

$$\boldsymbol{r}_k \equiv \boldsymbol{m}_k - \boldsymbol{h}_k(\boldsymbol{a}_k)$$
$$= (\mathbf{1} - \boldsymbol{H}_k \boldsymbol{K}_k)\, \boldsymbol{r}_k^{k-1}$$

- Covariance Matrix

$$\boldsymbol{R}_k = (\mathbf{1} - \boldsymbol{H}_k \boldsymbol{K}_k)\, \boldsymbol{V}$$
$$= \boldsymbol{V}_k \boxed{- \boldsymbol{H}_k \boldsymbol{C}_k \boldsymbol{H}_k^T}$$

Measurement Noise

Gain due to Information from previous measurements

- Chi Square Increment

$$\chi^2_+ = \boldsymbol{r}_k^T \boldsymbol{R}_k^{-1} \boldsymbol{r}_k$$
$$= \boldsymbol{r}_k^T \boldsymbol{G}_k \boldsymbol{r}_k + (\boldsymbol{a}_k - \boldsymbol{a}_k^{k-1})^T \left(\boldsymbol{C}_k^{k-1}\right)^{-1} (\boldsymbol{a}_k - \boldsymbol{a}_k^{k-1})$$

Smoothing

$$\{\boldsymbol{m}_{k'}; k' \leq k\} \mapsto \boldsymbol{a}_{k''<k} \quad : \text{ past}$$

- State Vector

New Pull

$$\boldsymbol{a}_k^n = \boldsymbol{a}_k + \boldsymbol{A}_k \left(\boldsymbol{a}_{k+1}^n - \boldsymbol{a}_{k+1}^k \right)$$

Smoothing Matrix

$$\boldsymbol{A}_k \equiv \boldsymbol{C}_k \boldsymbol{F}_k^T \left(\boldsymbol{C}_{k+1}^k \right)^{-1}$$

already calculated in the prediction step

already calculated in the prediction step

already calculated in the filtering step

It is instructive to compare filtering and smoothing formulae

Filter

$$\boldsymbol{a}_k = \boldsymbol{a}_k^{k-1} + \boldsymbol{K}_k \left(\boldsymbol{m}_k - \boldsymbol{h}_k(\boldsymbol{a}_k^{k-1})\right)$$

$$\boldsymbol{K}_k \equiv \boldsymbol{C}_k^{k-1}\boldsymbol{H}_k^T \left(\boldsymbol{R}_k^{k-1}\right)^{-1}$$

New Pull

Smoothing

$$\boldsymbol{A}_k \equiv \boldsymbol{C}_k \boldsymbol{F}_k^T \left(\boldsymbol{C}_{k+1}^k\right)^{-1}$$

$$\boldsymbol{a}_k^n = \boldsymbol{a}_k + \boldsymbol{A}_k(\boldsymbol{a}_{k+1}^n - \boldsymbol{a}_{k+1}^k)$$

- Covariance Matrix

Improvement from Measurements at $(k+1 \sim n)$

negative definite

$$C_k^n = \boxed{C_k} + A_k \left(\boxed{C_{k+1}^n - C_{k+1}^k} \right) A_k^T$$

already calculated in the prediction step

already calculated in the previous step

already calculated in the filtering step

- **Residual**

$$\begin{aligned} \boldsymbol{r}_k^n &\equiv \boldsymbol{m}_k - \boldsymbol{h}_k(\boldsymbol{a}_k^n) \\ &= \boldsymbol{r}_k - \boldsymbol{H}_k(\boldsymbol{a}_k^n - \boldsymbol{a}_k) \end{aligned}$$

- **Covariance Matrix**

$$\begin{aligned} \boldsymbol{R}_k^n &= \boldsymbol{R}_k - \boldsymbol{H}_k \boldsymbol{A}_k \left(\boldsymbol{C}_{k+1}^n - \boldsymbol{C}_{k+1}^k \right) \boldsymbol{A}_k^T \boldsymbol{H}_k^T \\ &= \boldsymbol{V}_k - \boldsymbol{H}_k \boldsymbol{C}_k^n \boldsymbol{H}_k^T \end{aligned}$$

Measurement Noise

Gain due to Information from other measurements

Inverse Kalman Filter

Machineary to eliminate measurement (k)

- State Vector

Pull we want to eliminate

$$\boldsymbol{a}_k^{n*} = \boldsymbol{a}_k^n + \boldsymbol{K}_k^{n*} \left(\boldsymbol{m}_k - \boldsymbol{h}_k(\boldsymbol{a}_k^n)\right)$$

Kalman Inverse Gain Matrix

already calculated in the prediction step

$$\boldsymbol{K}_k^{n*} \equiv \boldsymbol{C}_k^n \boldsymbol{H}_k^T \left(-\boldsymbol{V}_k + \boldsymbol{H}_k \boldsymbol{C}_k^n \boldsymbol{H}_k^T \right)^{-1}$$

already calculated in the smoothing step

- Covariance Matrix for state vector

$$
\begin{aligned}
\boldsymbol{C}_k^{n*} &= (\mathbf{1} - \boldsymbol{K}_k^{n*}\boldsymbol{H}_k)\,\boldsymbol{C}_k^n \\
&= \left[\left(\boxed{\boldsymbol{C}_k^n}\right)^{-1} - \boxed{\boldsymbol{H}_k^T}\,\boldsymbol{G}_k \boxed{\boldsymbol{H}_k} \right]^{-1}
\end{aligned}
$$

already calculated in the prediction step

already calculated in the smoothing step

- Covariance Matrix for residual

$$
\boldsymbol{R}_k^{n*} = \boldsymbol{V}_k + \boldsymbol{H}_k \boldsymbol{C}_k^{n*} \boldsymbol{H}_k^T
$$

Typical Usage of Kalman Filter in Tracking

Typical Procedure for Tracking

outermost innermost

$$(1) \to (2) \to \cdots \to (k-1) \to (k) \to \cdots (n)$$

filtering

$$\begin{tikzcd}
 & \boldsymbol{a}_2^1 \arrow[d] & & \boldsymbol{a}_{k-1}^{k-2} \arrow[d] & \boldsymbol{a}_k^{k-1} \arrow[d] & & \boldsymbol{a}_n \\
\boldsymbol{a}_1 \arrow[ur] & \boldsymbol{a}_2 \arrow[ur] & \cdots \arrow[ur] & \boldsymbol{a}_{k-1} \arrow[ur] & \boldsymbol{a}_k \arrow[ur] & \cdots \arrow[ur] &
\end{tikzcd}$$

prediction

$$(1) \cdots \leftarrow (k) \leftarrow (k+1) \leftarrow \cdots \leftarrow (n-1) \leftarrow (n)$$

$$\boldsymbol{a}_1^n \cdots \leftarrow \boldsymbol{a}_k^n \leftarrow \boldsymbol{a}_{k+1}^n \leftarrow \cdots \leftarrow \boldsymbol{a}_{n-1}^n \leftarrow \boldsymbol{a}_n \equiv \boldsymbol{a}_n^n$$

smoothing

Extrapolation

$$\boldsymbol{a}_n \xrightarrow[\text{prediction}]{} \boldsymbol{a}_{n+1}^n = \boldsymbol{a}_{IP} \qquad \boldsymbol{a}_0^n = \boldsymbol{a}_{\mathrm{cal}} \xleftarrow[\text{prediction}]{} \boldsymbol{a}_1^n$$

Alignment, Resolution Study, etc.

Need to eliminate point (k)

$$(1)\cdots\cdots\cdots(k-1) \quad (k) \quad (k+1)\cdots\cdots\cdots(n)$$

$\Downarrow$ Inverse Kalman Filter

$\boldsymbol{a}_k^{n*}$ Reference Track Param.

$\Downarrow$

$\boldsymbol{h}_k(\boldsymbol{a}_k^{n*})$ Expected Hit Position

$\Downarrow$

$\boldsymbol{r}_k^{n*} = \boldsymbol{m}_k - \boldsymbol{h}_k(\boldsymbol{a}_k^{n*})$ Residual to Look At

C++ Implementation Kalman Filter Library

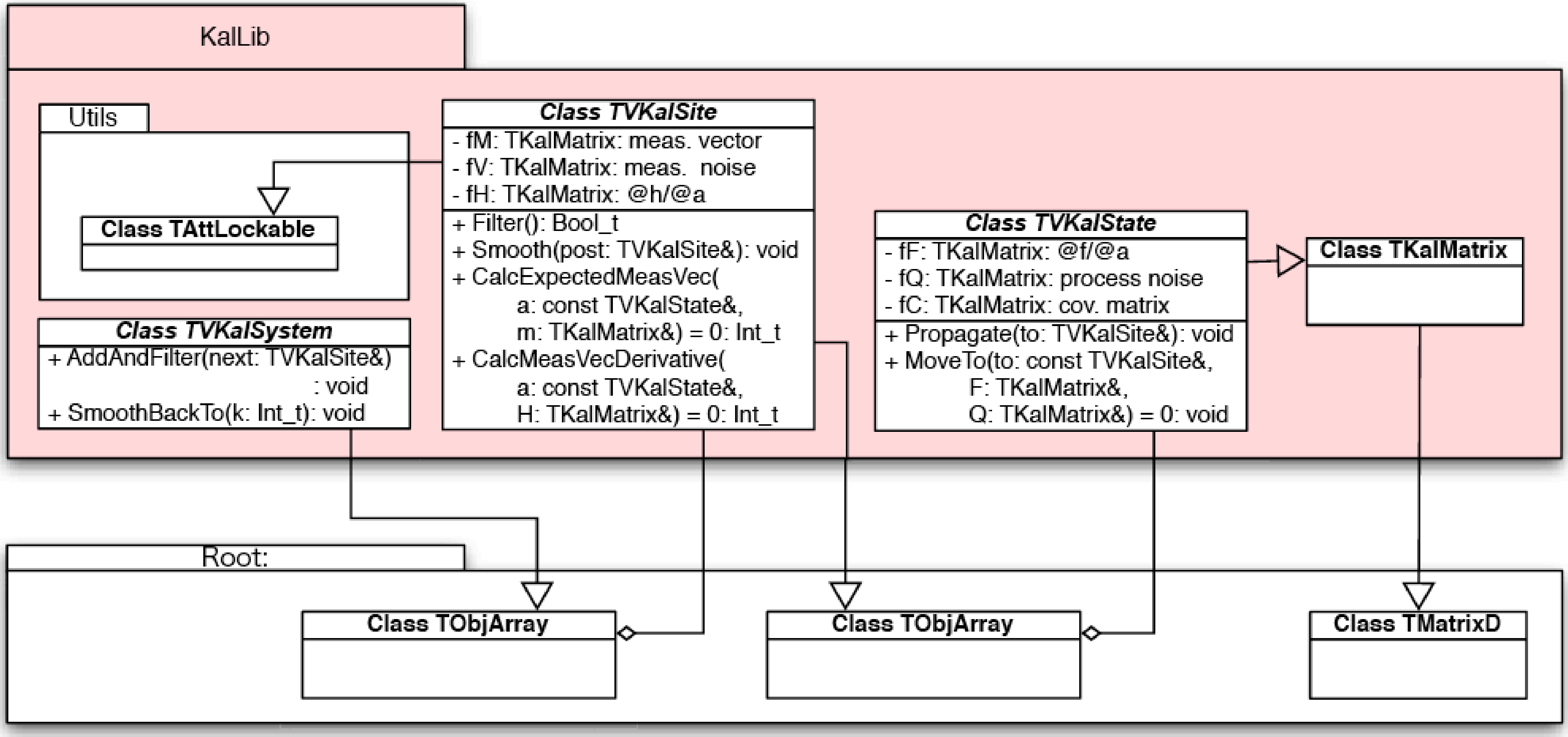
KF, Y.Nakashima, and A.Yamaguchi

Kalman Filter Library Features

- KalLib: general base classes that implement algorithm
  - TVKalSystem, TVKalSite, TVKalState
- KalTrackLib: that implements pure virtuals of KalLib for track fitting purpose
- GeomLib: geometry classes that provide
  - track models (helix, straight line, ...)
  - surfaces (cylinder, hyperboloid, flat plane, ...)

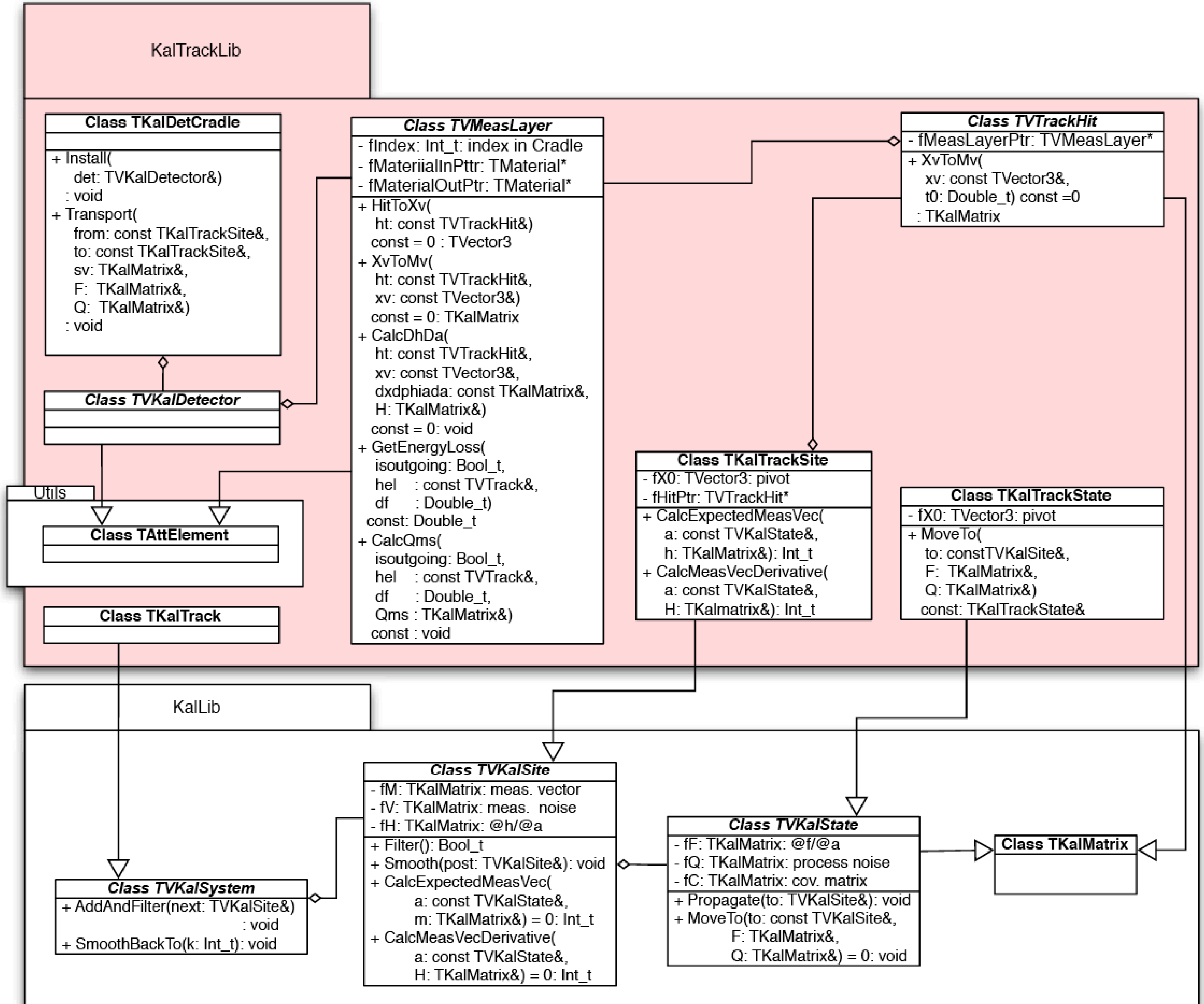
- Minimum number of user-implemented classes
  - MeasLayer : measurement layer
  - KalDetector : an array containing MeasLayers
    - You can put different kinds of MeasLayers
  - Hit : coordinate vector as defined by the MeasLayer
- Track model can change site to site which allows B-field variation along a particle trajectory

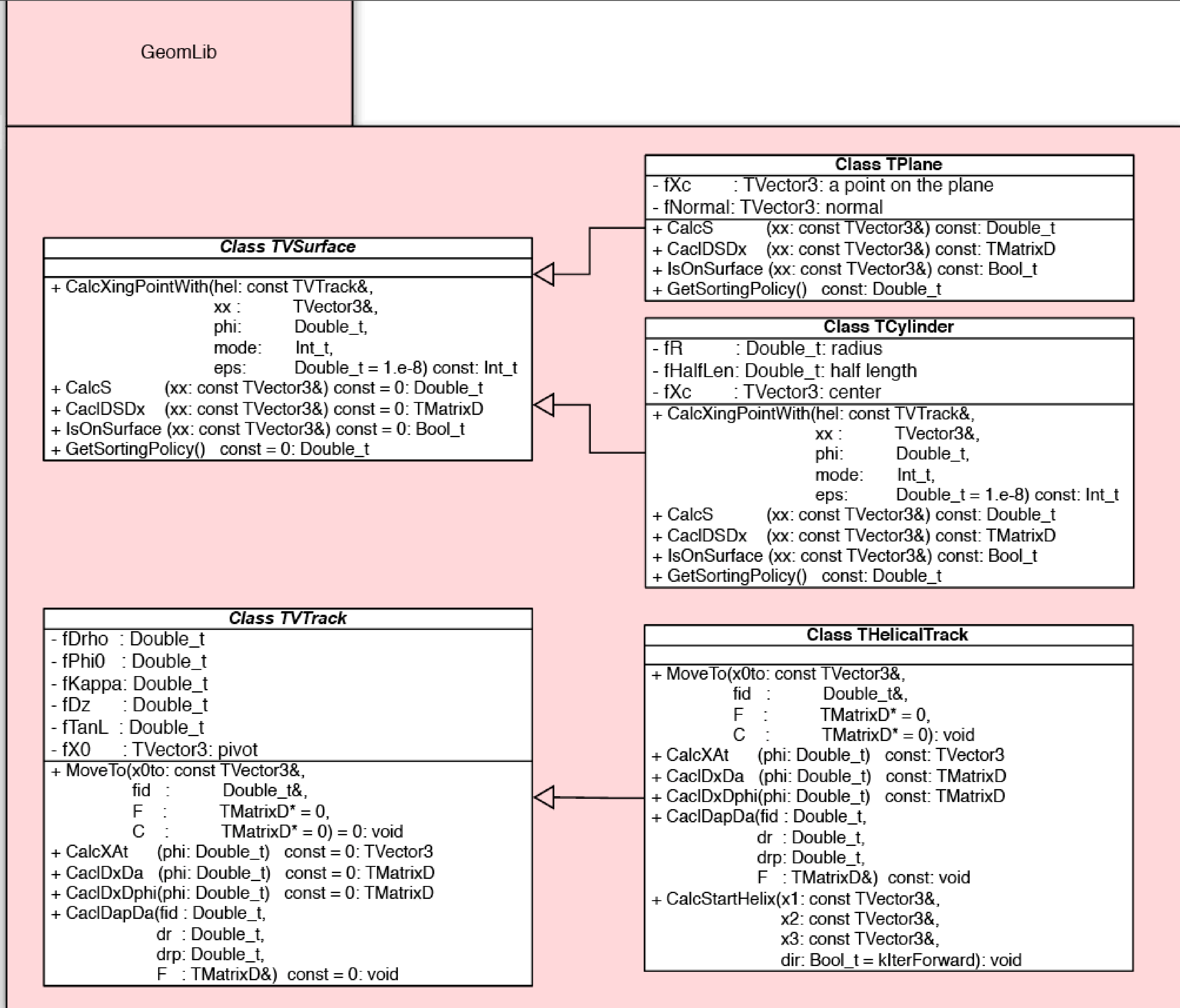
Kalman Filter Class Organization



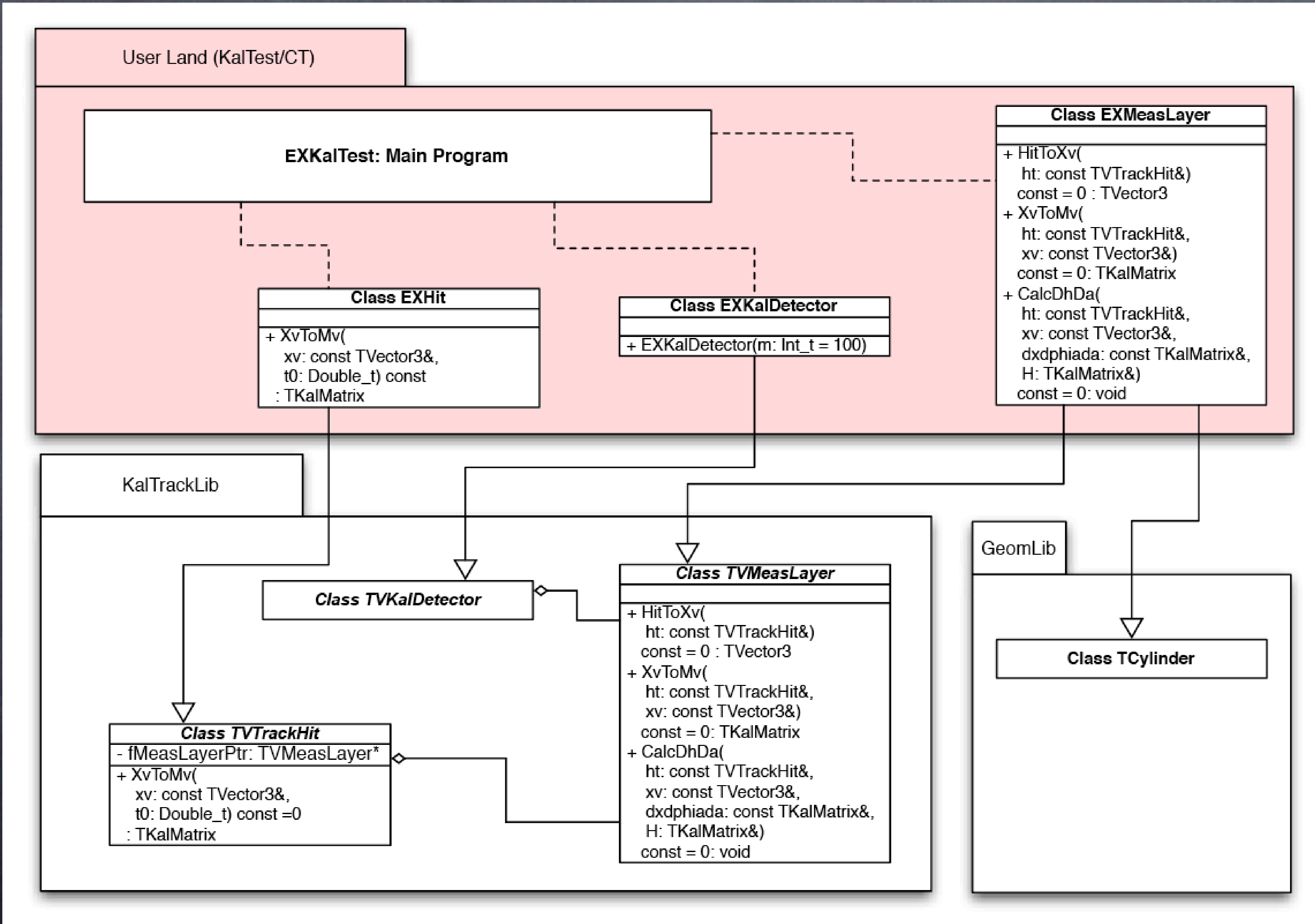
A TVKalSite carries predicted, filtered, and smoothed TVKalState's

Application-specific functions are pure virtual and to be implemented in a derived class


KalTrackLib
Class TKalDetCradle
+ Install(
det: TVKalDetector&)
: void
+ Transport(
from: const TKalTrackSite&,
to: const TKalTrackSite&,
sv: TKalMatrix&,
F: TKalMatrix&,
Q: TKalMatrix&)
: void
Class TVKalDetector
Utils
Class TAttElement
Class TKalTrack
Class TVMeasLayer
- fIndex: Int_t: index in Cradle
- fMateriialInPttr: TMaterial*
- fMaterialOutPtr: TMaterial*
+ HitToXv(
ht: const TVTrackHit&)
const = 0 : TVector3
+ XvToMv(
ht: const TVTrackHit&,
xv: const TVector3&)
const = 0: TKalMatrix
+ CalcDhDa(
ht: const TVTrackHit&,
xv: const TVector3&,
dxdphiada: const TKalMatrix&,
H: TKalMatrix&)
const = 0: void
+ GetEnergyLoss(
isoutgoing: Bool_t,
hel : const TVTrack&,
df : Double_t)
const: Double_t
+ CalcQms(
isoutgoing: Bool_t,
hel : const TVTrack&,
df : Double_t,
Qms : TKalMatrix&)
const : void
Class TVTrackHit
- fMeasLayerPtr: TVMeasLayer*
+ XvToMv(
xv: const TVector3&,
t0: Double_t) const =0
: TKalMatrix
Class TKalTrackSite
- fX0: TVector3: pivot
- fHitPtr: TVTrackHit*
+ CalcExpectedMeasVec(
a: const TVKalState&,
h: TKalMatrix&): Int_t
+ CalcMeasVecDerivative(
a: const TVKalState&,
H: TKalmatrix&): Int_t
Class TKalTrackState
- fX0: TVector3: pivot
+ MoveTo(
to: constTVKalSite&,
F: TKalMatrix&,
Q: TKalMatrix&)
const: TKalTrackState&
KalLib
Class TVKalSite
- fM: TKalMatrix: meas. vector
- fV: TKalMatrix: meas. noise
- fH: TKalMatrix: @h/@a
+ Filter(): Bool_t
+ Smooth(post: TVKalSite&): void
+ CalcExpectedMeasVec(
a: const TVKalState&,
m: TKalMatrix&) = 0: Int_t
+ CalcMeasVecDerivative(
a: const TVKalState&,
H: TKalMatrix&) = 0: Int_t
Class TVKalSystem
+ AddAndFilter(next: TVKalSite&)
: void
+ SmoothBackTo(k: Int_t): void
Class TVKalState
- fF: TKalMatrix: @f/@a
- fQ: TKalMatrix: process noise
- fC: TKalMatrix: cov. matrix
+ Propagate(to: TVKalSite&): void
+ MoveTo(to: const TVKalSite&,
F: TKalMatrix&,
Q: TKalMatrix&) = 0: void
Class TKalMatrix


GeomLib
Class TPlane
- fXc : TVector3: a point on the plane
- fNormal: TVector3: normal
+ CalcS (xx: const TVector3&) const: Double_t
+ CaclDSDx (xx: const TVector3&) const: TMatrixD
+ IsOnSurface (xx: const TVector3&) const: Bool_t
+ GetSortingPolicy() const: Double_t
Class TVSurface
+ CalcXingPointWith(hel: const TVTrack&,
xx : TVector3&,
phi: Double_t,
mode: Int_t,
eps: Double_t = 1.e-8) const: Int_t
+ CalcS (xx: const TVector3&) const = 0: Double_t
+ CaclDSDx (xx: const TVector3&) const = 0: TMatrixD
+ IsOnSurface (xx: const TVector3&) const = 0: Bool_t
+ GetSortingPolicy() const = 0: Double_t
Class TCylinder
- fR : Double_t: radius
- fHalfLen: Double_t: half length
- fXc : TVector3: center
+ CalcXingPointWith(hel: const TVTrack&,
xx : TVector3&,
phi: Double_t,
mode: Int_t,
eps: Double_t = 1.e-8) const: Int_t
+ CalcS (xx: const TVector3&) const: Double_t
+ CaclDSDx (xx: const TVector3&) const: TMatrixD
+ IsOnSurface (xx: const TVector3&) const: Bool_t
+ GetSortingPolicy() const: Double_t
Class TVTrack
- fDrho : Double_t
- fPhi0 : Double_t
- fKappa: Double_t
- fDz : Double_t
- fTanL : Double_t
- fX0 : TVector3: pivot
+ MoveTo(x0to: const TVector3&,
fid : Double_t&,
F : TMatrixD* = 0,
C : TMatrixD* = 0) = 0: void
+ CalcXAt (phi: Double_t) const = 0: TVector3
+ CaclDxDa (phi: Double_t) const = 0: TMatrixD
+ CaclDxDphi(phi: Double_t) const = 0: TMatrixD
+ CaclDapDa(fid : Double_t,
dr : Double_t,
drp: Double_t,
F : TMatrixD&) const = 0: void
Class THelicalTrack
+ MoveTo(x0to: const TVector3&,
fid : Double_t&,
F : TMatrixD* = 0,
C : TMatrixD* = 0): void
+ CalcXAt (phi: Double_t) const: TVector3
+ CaclDxDa (phi: Double_t) const: TMatrixD
+ CaclDxDphi(phi: Double_t) const: TMatrixD
+ CaclDapDa(fid : Double_t,
dr : Double_t,
drp: Double_t,
F : TMatrixD&) const: void
+ CalcStartHelix(x1: const TVector3&,
x2: const TVector3&,
x3: const TVector3&,
dir: Bool_t = kIterForward): void

Sample User Program

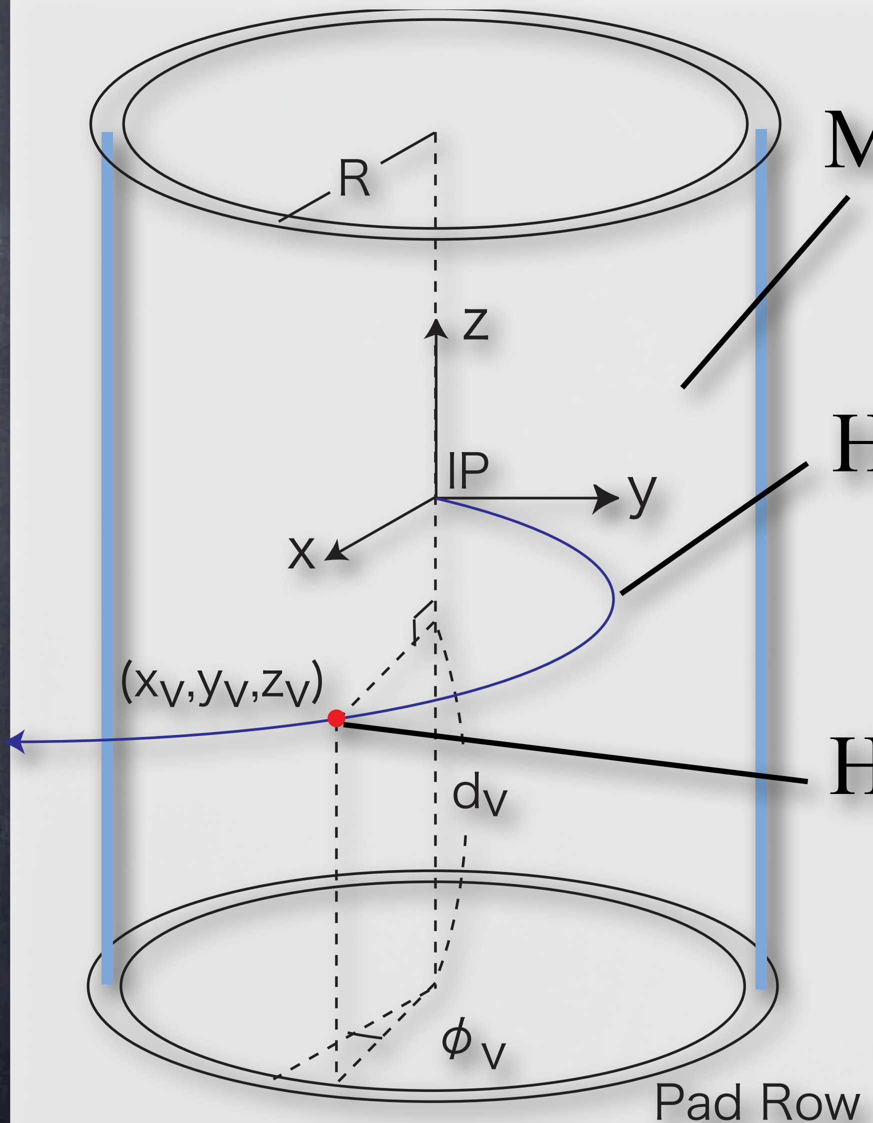


Application to ILC Track Fitting

Example of Detector Implementation

TPC Implementation

Define a KalDetector (TPCKalDetector) inheriting TVKalDetector



Measurement Layer

$$
\begin{aligned}
S(x,y,z) &= x^2 + y^2 - R^2 \\
&= 0
\end{aligned}
$$

Helix

$$
\left\{
\begin{aligned}
x &= x_0 + d_\rho \cos\phi_0 + \tfrac{\alpha}{\kappa}\left(\cos\phi_0 - \cos(\phi_o + \phi)\right) \\
y &= y_0 + d_\rho \sin\phi_0 + \tfrac{\alpha}{\kappa}\left(\sin\phi_0 - \sin(\phi_o + \phi)\right) \\
z &= z_0 + d_z \qquad\qquad - \tfrac{\alpha}{\kappa}\tan\lambda \cdot \phi
\end{aligned}
\right.
$$

Hit

$$
\begin{pmatrix} R \cdot \phi_v \\ d_v \end{pmatrix} = \begin{pmatrix} R \cdot \tan^{-1}\left(\dfrac{y_v}{x_v}\right) \\ L/2 \mp z_v \end{pmatrix}
$$

Define TPCMeasLayer by inheriting TVMeasLayer and implement its pure virtual methods:

HitToXv

$$\begin{pmatrix} x_v \\ y_v \\ z_v \end{pmatrix} = \begin{pmatrix} R \cdot \cos\phi_v \\ R \cdot \sin\phi_v \\ \pm(L/2 - d_v) \end{pmatrix}$$

XvToMv

$$\begin{pmatrix} R \cdot \phi_v \\ d_v \end{pmatrix} = \begin{pmatrix} R \cdot \tan^{-1}\left(\frac{y_v}{x_v}\right) \\ L/2 \mp z_v \end{pmatrix}$$

CalcDhDa

Meas.Vector Derivative w.r.t. Track Parameter Vector

$$\begin{pmatrix} \frac{\partial(R\cdot\phi_v)}{\partial \mathrm{a}} \\ \frac{\partial d_v}{\partial \mathrm{a}} \end{pmatrix} = \begin{pmatrix} -\frac{y_v}{R}\left(\frac{\partial x_v}{\partial \mathrm{a}}\right) + \frac{x_v}{R}\left(\frac{\partial y_v}{\partial \mathrm{a}}\right) \\ \mp \frac{\partial z_v}{\partial \mathrm{a}} \end{pmatrix}$$

$$\frac{\partial \mathrm{x}(\phi(\mathrm{a}),\mathrm{a})}{\partial \mathrm{a}} = \frac{\partial \mathrm{x}}{\partial \phi} \cdot \frac{\partial \phi}{\partial \mathrm{a}} + \frac{\partial \mathrm{x}}{\partial \mathrm{a}}$$

$$\frac{\partial \phi}{\partial \mathrm{a}} = -\frac{1}{\left(\frac{\partial S}{\partial \mathrm{X}} \cdot \frac{\partial \mathrm{X}}{\partial \phi}\right)} \frac{\partial S}{\partial \mathrm{X}} \cdot \frac{\partial \mathrm{X}}{\partial \mathrm{a}}$$

Notice that some TPCMeasLayer's may be implemented as dummy representing just boundaries of different materials

Add TPCMeasLayer's to TPCKalDetector with Add(..) method to complete TPC implementation

Integration of different trackers into a single tracking system

Define other trackers such as IT and VTX in a similar way and install them into TKalDetCradle as

```
TKalDetCradle   toygld;
VTXKalDetector  vtxdet;  toygld.Indtall(vtxdet);
ITKalDetector   itdet;   toygld.Install(itdet);
TPCKalDetector  tpcdet;  toygld.Install(tpcdet);
toygld.Sort();
```

Upon installation of each detector, its shell evaporates and only its MeasLayer's remain flatly expanded in the cradle

The last line sorts out the flatly expanded MeasLayer's from inside to outside

```
// ---------------------------
//  Add sited to the kaltrack
// ---------------------------

EXHYBTrack kaltrack;   // a track is a kal system
kaltrack.SetOwner();   // kaltrack owns sites
kaltrack.Add(&sited);  // add the dummy site to this track

// ---------------------------
//  Prepare hit iterrator
// ---------------------------

TIter nextsite(&kalhits, gkDir); // come in to IP, if gkDir = kIterBackward

// ---------------------------
//  Start Kalman Filter
// ---------------------------

TVTrackHit *hitp = 0;
while ((hitp = dynamic_cast<TVTrackHit *>(nextsite()))) {
  TKalTrackSite  &site = *new TKalTrackSite(*hitp); // new site
  if (!kaltrack.AddAndFilter(site)) {               // filter it
    cerr << " site discarded!" << endl;
    delete &site;                                   // delete it if failed
  }
} // end of Kalman filter

// ---------------------------
//  Smooth the track
// ---------------------------

kaltrack.SmoothBackTo(0);
```

More information available from the following URL:

http://www-jlc.kek.jp/subg/offl/kaltest/

where you can find a reference manual for the KalTest package and some other useful documents.

The reference manual contains full derivations of relevant formulae for extended Kalman filter technique.